Personal and Password-Based Cryptography

Vom Fachbereich Informatik (FB 20) der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doctor rerum naturalium (Dr. rer. nat.)

von
Dipl.-Inf. (FH) Kai Wilhelm Samelin, M. Sc. M. Sc. M. Sc.
geboren in Haan

Referenten:
Prof. Dr. Michael Waidner (Referent)
Prof. Dr. Colin Boyd (Korreferent)
Dr. Jan Camenisch (Korreferent)

Tag der Einreichung: 16.01.2018
Tag der Disputation: 23.02.2018

Hochschulkennziffer: D 17

Zürich, 2018

Verfasser: Kai Samelin
Titel: Personal and Password-Based Cryptography
Dissertationsort: Darmstadt, Technische Universität Darmstadt
Veröffentlichungsjahr der Dissertation auf TUprints: 2018
Tag der mündlichen Prüfung: 23.02.2018

PhD Referees:
Prof. Dr. Michael Waidner
Prof. Dr. Colin Boyd
Dr. Jan Camenisch

Further PhD Committee Members:
Prof. Dr. Felix Wolf (Chair)
Prof. Dr. Sebastian Faust
Prof. Dr. Kristian Kersting

Erklärung

Hiermit erkläre ich, diese Arbeit selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben.

Hiermit erkläre ich weiterhin, dass von mir bisher kein Promotionsversuch unternommen wurde.

Kai W. Samelin
Zürich, 2018

Acknowledgements

First of all, I want to express my gratitude to my supervisor at IBM Research – Zurich, Jan Camenisch. He is an humongous teacher with a great attitude, mind-blowing knowledge and an awesome way of teaching. Moreover, he is also one of the kindest persons I was allowed to meet in life. It was a really nice experience working with him and his group, as he was always willing to share his knowledge without hesitation, while also keeping me (at least most of the time) focused on the important stuff. Thank you for all the nice discussions!

I also owe Michael Waidner, my advisor in Darmstadt, a lot. I am truly in debt that he accepted me as his PhD-student at the Technical University of Darmstadt. This also includes Colin Boyd which agreed to join the referees, as well as my graduation committee members Sebastian Faust, Kristian Kersting and Felix Wolf.

I would also like to thank my colleagues at IBM Research – Zurich: Costas Bekas, Patrik Bichsel, Sören Bleikertz, Cecilia Boschini, Marcus Brandenburger, Christian Cachin, Manu Drijvers, Maria Dubovitskaya, Robert Enderlein, Eduarda Freire, Alexandra Gorski, Bernhard Jansen, Dániel Kovács, Stephan Krenn, Anja Lehmann, Zoltán Nagy, Gregory Neven, Michael Osbourne, Franz-Stefan Preiss, Dieter Sommer, Björn Tackmann, Axel Tanner and Hagen Völzer. They made my stay in Switzerland even more enjoyable.

My thanks also include my prior colleagues in Darmstadt and Passau: Steven Arzt, Robert Basmadjian, Andreas Berl, Renate Birmelin, Kevin Falzon, Andreas Fischer, Andreas Follner, Tommaso Gagliardoni, Xiaobing He, Ralph Herkenhöner, Richard Holzer, Anna Kaplan, Fabian Kokot, Robert Künnemann, Silvia Lehmbeck, Elisabeth Loibl, Gergő Lovász, Florian Niedermeier, Michael Niedermeier, Siegfried Rasthofer, Thomas Schneider, Patrick Wüchner, Sui Zhiyuan and Michael Zohner. It was always fun with them, and sometimes even some good ideas were discussed.

It was also a great honor to work with some of the smartest people I met so far, i.e., my co-authors: Foteini Baldimtsi, Michael Till Beck, Arne Bilzhause, Juan Felipe Botero, Chris Brzuska, David Derler, Christoph Frädrich, Yossi Gilad, Focke Höhne, Manuel Huber, Anna Lysyanskaya, Hermann de Meer, Henrich C. Pöhls, Wolfgang Popp, Joachim Posegga, Stefan Peters, Leonid Reyzin, Daniel Slamanig and Sophia Yakoubov. Thanks a lot for the discussions, help and insights!

Of course, I want to also thank my friends and family. In particular, my mother, my grandparents and my step-father and everyone I may have forgotten. They supported me throughout everything so far. Thanks for everything!

Kai W. Samelin
Zürich, 2018

Publications

The following papers have been published prior to submission of this thesis or are currently under submission.

Publications Part of this Thesis

The following list of papers found their way into this thesis.

1. J. Camenisch, Y. Gilad, A. Lehmann, Z. A. Nagy, G. Neven, and K. Samelin. *UC-Secure Distributed Password-Based Single Sign-On.* Under Submission.
2. J. Camenisch, A. Lehmann, G. Neven, and K. Samelin. *Virtual Smart Cards: How to Sign with a Password and a Server.* In Proceedings of the 10th Conference on Security and Cryptography for Networks (SCN'16), Springer LNCS 9841, 2016, pages 353-371.
3. J. Camenisch, A. Lehmann, G. Neven, and K. Samelin. *UC-Secure Non-Interactive Public-Key Encryption.* In Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17), IEEE, 2017, pages 217–233.
4. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. *Chameleon-Hashes with Ephemeral Trapdoors And Applications to Invisible Sanitizable Signatures.* In Proceedings of the 20th International Conference on Practice and Theory in Public-Key Cryptography (PKC'17, Part II), Springer LNCS 10175 2017, pages 152–182.
5. M. T. Beck, S. Krenn, F.-S. Preiss, and K. Samelin. *Practical Signing-Right Revocation.* In Proceedings of the 9th International Conference on Trust & Trustworthy Computing (TRUST'16), Springer LNCS 9824, 2016, pages 21-39.

Other Publications

These are the other publications which have not found their way into this thesis.

Journals and Invited Papers

1. H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin *Redactable Signature Schemes for Trees With Signer-Controlled Non-Leaf-Redactions.* Springer CCIS 455, 2014, pages 155-171.
2. F. Höhne, H. C. Pöhls, and K. Samelin *Rechtsfolgen editierbarer Signaturen.* Datenschutz und Datenrecht (DuD), Volume 36 (6): 485-491, 2012.

Conferences and Workshops

1. S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. *Chameleon-Hashes with Dual Long-Term Trapdoors and Their Applications.* In Proceedings of the 10th International Conference on Cryptology (AfricaCrypt'18), Springer LNCS 10831, 2018, pages 11-32.
2. A. Bilzhause, H. C. Pöhls, and K. Samelin. *Position Paper: The Past, Present, and Future of Sanitizable and Redactable Signatures.* In Proceedings of the 2nd Workshop on Security, Privacy, and Identity Management in the Cloud (SECPID@ARES'17), ACM, 2017, pages 87:1-87:9.
3. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. *Practical Strongly Invisible and Strongly Accountable Sanitizable Signatures.* In Proceedings of the 22nd Australasian Conference on Information Security and Privacy (ACISP'17, Part I), Springer LNCS 10342, 2017, pages 437-452.
4. F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov. *Efficient Accumulators with Applications to Anonymity-Preserving Revocation.* In Proceedings of the 2nd IEEE European Symposium on Security and Privacy (IEEE EuroS&P'17), IEEE, 2017, pages 301-315.
5. C. Frädrich, H. C. Pöhls, W. Popp, N. Rakotondravony, and K. Samelin. *Integrity and Authenticity Protection with Selective Disclosure Control in the Cloud & IoT.* In Proceedings of the 18th International Conference on Information and Communications Security (ICICS'16), Springer LNCS 9977, 2016, pages 197-213.
6. M. T. Beck, J. F. Botero, and K. Samelin. *Resilient Allocation of Service Function Chains.* In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Network (IEEE NFV-SDN'16), IEEE, 2016, pages 128-133.
7. A. Bilzhause, M. Huber, H. C. Pöhls, and K. Samelin. *Cryptographically Enforced Four-Eyes Principle.* In Proceedings of the 1st Workshop on Security, Privacy, and Identity Management in the Cloud (SECPID@ARES'16), IEEE, 2016, pages 760-767.
8. D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. *A General Framework for Redactable Signatures and New Constructions.* In Proceedings of the 18th Annual International Conference on Information Security and Cryptology (ICISC'15), Springer LNCS 9558, 2015, pages 3-19.
9. S. Krenn, K. Samelin, and D. Sommer. *Stronger Security for Sanitizable Signatures.* In Proceedings of the 10th International Workshop on Data Privacy Management, and Security Assurance (DPM@ESORICS'15), Springer LNCS 9481, 2015, pages 100-117.
10. Henrich C. Pöhls, and K. Samelin *Accountable Redactable Signatures.* In Proceedings of the 10th International Conference on Availability, Reliability and Security (ARES'15), IEEE, 2015, pages 60-69.
11. H. C. Pöhls, and K. Samelin *On Updatable Redactable Signatures.* In Proceedings of the 12th International Conference on Applied Cryptography and Network Security (ACNS'14), Springer LNCS 8479, 2014, pages 457-475.
12. H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. *On the Relation Between Redactable and Sanitizable Signatures Schemes.* In Proceedings of the 6th International Symposium on Engineering Secure Software and Systems (ESSoS'14), Springer LNCS 8364, 2014, pages 113-130.

13. C. Brzuska, H. C. Pöhls, and K. Samelin *Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures.* In Proceedings of the 10th European PKI Workshop: Research and Applications (EuroPKI@ESORICS'2013), Springer LNCS 8341, 2013, pages 12-30.
14. H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. *Scope of Security Properties of Sanitizable Signatures Revisited.* In Proceedings of the 8th International Conference on Availability, Reliability and Security (ARES'13), IEEE, 2013, pages 188-197.
15. H. C. Pöhls, S. Peters, K. Samelin, J. Posegga. and H. de Meer. *Malleable Signatures for Resource Constrained Platforms.* In Proceedings of the 7th IFIP WG 11.2 International Workshopth International Conference on Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems (WISTP'2013), Springer LNCS 7886, 2013, pages 18-33.
16. C. Brzuska, H. C. Pöhls, and K. Samelin. *Non-Interactive Public Accountability for Sanitizable Signatures.* In Proceedings of the 9th European PKI Workshop: Research and Applications (EuroPKI@ESORICS'12), Springer LNCS 7868, 2012, pages 178-193.
17. H. C. Pöhls, K. Samelin, H. de Meer, and J. Posegga. *Flexible Redactable Signature Schemes for Trees: Extended Model and Construction.* In Proceedings of the 8th International Conference on Security and Cryptography (SECRYPT'12), SciTePress, 2012, pages 113-125.
18. K. Samelin, H. C. Pöhls, A. Bilzhause, and J. Posegga. *On Structural Signatures for Tree Data Structures.* In Proceedings of the 10th International Conference on Applied Cryptography and Network Security (ACNS'12), Springer LNCS 7341, 2012, pages 171-187.
19. K. Samelin, H. C. Pöhls, A. Bilzhause, and J. Posegga. *Redactable Signatures for Independent Removal of Structure and Content.* In Proceedings of the 8th International Conference on Information Security Practice and Experience (ISPEC'12), Springer LNCS 7232, 2012, pages 17-33.
20. H. C. Pöhls, K. Samelin, and J. Posegga. *Sanitizable Signatures in XML Signature — Performance, Mixing Properties, and Revisiting the Property of Transparency.* In Proceedings of the 9th International Conference on Applied Cryptography and Network Security (ACNS'11), Springer LNCS 6715, 2011, pages 166-182.
21. H. C. Pöhls, A. Bilzhause, K. Samelin, and J. Posegga. *Sanitizable Signed Privacy Preferences for Social Networks.* In Proceedings of GI Workshop on Privacy and Identity Management for Communities - Communities for Privacy and Identity Management (DICCDI'11), GI, 2011, pages 409-424.

Technical Reports

1. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig *Practical Strongly Invisible and Strongly Accountable Sanitizable Signatures*, IACR ePrint Archive 2017/445, IACR, 2017.
2. F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov. *Efficient Accumulators with Applications to Anonymity-Preserving Revocation.* IACR ePrint Archive 2017/043, IACR, 2017.
3. M. Brandenburger, J. Camenisch, and K. Samelin. *Method for Privacy-Preserving Enforcing Non-Hackable Software Licensing with CPU-Enclaves and User-Controlled In- and Output*

Observability IPCOM000249237, IP.com, 2017.

4. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. *Chameleon-Hashes with Ephemeral Trapdoors And Applications to Invisible Sanitizable Signatures.* IACR ePrint Archive 2017/011, IACR, 2017.
5. J. Camenisch, K. Samelin, D. Sommer, and H. Völzer. *Method for Automatically Deriving the Privacy Policy for a Service from a Set of Process Models.* IPCOM000245320D, IP.com, 2016.
6. J. Camenisch, A. Lehmann, G. Neven, and K. Samelin. *Virtual Smart Cards: How to Sign with a Password and a Server.* IACR ePrint Archive 2015/1101, IACR, 2015.
7. D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. *A General Framework for Redactable Signatures and New Constructions.* IACR ePrint Archive 2015/1059, IACR, 2015.
8. H. de Meer, M. Liedel, H. C. Pöhls, J. Posegga, and K. Samelin. *Indistinguishability of One-Way Accumulators.* Technical Report MIP-1210, Universität Passau, 2012.
9. K. Samelin, H. C. Pöhls, J. Posegga, and H. de Meer. *Blockwise Accountability for Sanitizable Signatures.* Technical Report MIP-1208, Universität Passau, 2012.
10. H. C. Pöhls, K. Samelin, J. Posegga, and H. de Meer. *Transparent Mergeable Redactable Signatures with Signer Commitment and Applications.* Technical Report MIP-1206, Universität Passau, 2012.
11. H. C. Pöhls, K. Samelin, J. Posegga, and H. de Meer. *Length-Hiding Redactable Signatures from One-Way Accumulators in $\mathcal{O}(n)$.* Technical Report MIP-1201, Universität Passau, 2012.

Filed Patents

1. J. Camenisch, and K. Samelin. *Communication of Messages over Networks.*
2. J. Camenisch, M. Drijvers, A. Lehmann, and K. Samelin. *A Method for Leakge-Resilient Distributed Function Evaluation with CPU-Enclaves.*

Patent Applications

1. J. Camenisch, A. Lehmann, G. Neven, and K. Samelin. *Encrypted Message Authentication. US Patent App. 14/872,695.*
2. J. Camenisch, A. Lehmann, G. Neven, F.-S. Preiss, and K. Samelin. *Server-Assisted Authentication. US Patent App. 14/966,061.*
3. M. Brandenburger, F.-S. Preiss, K. Samelin, and D. Sommer. *Revocable PKI-Signatures. US Patent App. 15/146,707.*
4. D. Kovacs, K. Samelin, and D. Sommer. *Authentication via Revocable Signatures. US Patent App. 15/190,731.*
5. J. Camenisch, D. Kovacs, K. Samelin, and D. Sommer. *Credential-Based Authorization. US Patent App. 15/177,391.*

Granted Patents

1. J. Camenisch, F.-S. Preiss, K. Samelin, and D. Sommer. *System and Method for Generating a Server-Assisted Strong Password from a Weak Secret. US Patent 9,565,020.*

Wissenschaftlicher Werdegang

The author's CV has been removed for the purpose of data protection, as recommended for the electronic publication of dissertations at TU Darmstadt.

Abstract

This thesis addresses the question what cryptography can do for one personally, i.e., it looks at security and privacy challenges of individuals in today's world. In particular, this thesis solves a number of real-world problems, including secure handling of passwords used for authentication and how to extend digital signature schemes to allow for additional features. The presented protocols are provably secure under realistic assumptions, while providing state-of-the-art security and privacy guarantees. All proposed protocols are highly efficient, useful, yet deployable on a large scale, i.e., they are truly practical, thus bridging the gap between theory and practice. This is demonstrated by providing performance evaluations and estimates of selected protocols.

In more detail, this thesis is split up into two main parts. The first part of this thesis deals with protocols which allow a user to authenticate securely with a, potentially low-entropy, password which must be considered a valuable asset not to be made public. The second part applies several of the ideas given in the first part of this thesis to digital signatures. In particular, the ideas introduced add new possibilities and privacy features to this already very versatile primitive.

The first part of this thesis on protocols is split up into three sub-parts. The first sub-part addresses single sign-on (SSO) protocols. In existing work, ticket-granting server(s) can, e.g., impersonate users towards service providers or offline attack their passwords. To tackle this situation, two distributed password-based single sign-on (SSO) functionalities and their realizing protocols are presented, where the password check and token generation is distributed among multiple entities. Both functionalities are formulated in the universal-composition (UC) framework. This guarantees security in arbitrary contexts, while also absorbing unavoidable practical limitations such as typos, correlated password attempts by users and the case of guessed passwords into the definition. The first protocol offers the basic functionality one expects from such a distributed password-based SSO protocol, while the second protocol provides even more privacy guarantees. For example, the service providers no longer learn which other access rights an entity has, how long a token is valid and allows to establish different identities, i.e., pseudonyms, with each service provider.

The second sub-part introduces password-authenticated signatures, realizing virtual smart-cards, as real smart-cards have a number of serious drawbacks. For example, special smart-card readers are needed for usage and are not always available, while assuming that users always carry such readers with them is unrealistic. Virtual smart-cards circumvent these limitations by letting a user enter a password on a personal device, such as a smart-phone, to generate signatures on arbitrary messages with the help of an additional server. This approach prevents an adversary from using the signing key, if a user loses a device without also entering the correct password. The server only contributes to signature generation, if the password entered was correct. Neither the server nor the device alone can mount attacks on the password or on the

password attempts, while the server does not learn the messages signed. As for SSO, security is defined by providing an ideal functionality in the UC-framework, implying the same advantages. The realizing protocol is secure against adaptive adversaries, i.e., an adversary can adaptively corrupt any protocol participants. To account for the main use-case of lost devices, a new corruption model is introduced. Namely, the simulator does not receive all prior input and output upon corruption, which is necessary to model the case of lost devices such that the adversary does not receive the prior password attempts. This is accompanied by a new non-committing encryption scheme for the receiver which requires secure erasures. The implementation of the given protocol shows that it even outperforms state-of-the-art smart-cards.

In the third sub-part, a fully simulatable non-committing encryption scheme is introduced. In particular, the encryption scheme introduced for the virtual smart-cards requires secure erasures. However, this is not always a reasonable assumption. To tackle this situation, this part presents an extended definition and protocol which allows simulating non-interactive ciphertexts even without secure erasures in a fully adaptive way. Hence, the simulator can give away the randomness for secret key generation and the randomness used for ciphertext generation to an adaptive adversary simultaneously. Such a non-interactive definition is in particular useful, if ciphertexts are further processed. This is demonstrated by providing the first definition of UC-secure signcryption in a setting with adaptive corruptions without secure erasures, which was not possible before. However, this part also comes with an impossibility result: it is proven that neither such an encryption scheme nor signcryption can be realized in non-idealized models.

The second part of this thesis deals with digital signature schemes with additional features. Here, two main contributions are presented. The first contribution of this part is about sanitizable signature schemes. In already existing definitions of sanitizable signature schemes, a semi-trusted third party, named the sanitizer, can alter signer-chosen blocks of signed messages, but a third party can derive which parts are actually admissible. The newly introduced notion of invisible sanitizable signature schemes improves on this situation by also hiding which parts of a given message are sanitizable, adding an additional layer of privacy. To build this new primitive, the new notion of chameleon-hashes with ephemeral trapdoors is introduced. These chameleon-hashes allow one to find arbitrary collisions of a hash, if two trapdoors at the same time are known. One trapdoor is a long-term secret, while the second one is generated at hash generation.

Finally, this thesis address the case of signing-right revocation. Nowadays, a certificate needs to be checked whether it is revoked at every signature verification. As verification naturally occurs more often, this negatively impacts on practicality, as thus network connectivity at verification is required. The protocols presented solve this by letting the signature itself vouch for the fact that the certificate was not revoked at signature generation time. This is achieved by letting a revocation authority contribute to signature generation. To account for privacy concerns, the authority does not learn the messages signed, while an extension also prohibits that the authority can link a signing protocol to the final signature.

Summarized, this thesis presents provably secure protocols which are geared to be highly efficient and are of direct practical relevance for personal usage, meaning that the primitives can directly be deployed and used, even in today's infrastructure.

Kurzfassung

Diese Dissertation geht der Frage nach, wie Kryptographie alltägliche Probleme lösen kann. Genauer werden eine Reihe von Herausforderungen gelöst, die in der heutigen vernetzten Welt für dessen Benutzer von Bedeutung sind. Insbesondere wird in dieser Arbeit die sichere Handhabung von Benutzerpasswörtern und das Erweitern von digitalen Signaturen um zusätzliche Funktionen und Möglichkeiten diskutiert. Alle vorgestellten Protokolle sind unter Standardannahmen beweisbar sicher, geben Benutzern weitgehende Privatheit- und Sicherheitsgarantien, während sie gleichzeitig sehr effizient und direkt benutzbar sind. Dies wird dadurch demonstriert, dass einige ausgewählte Protokolle implementiert bzw. Laufzeitabschätzungen präsentiert werden.

Genauer ist diese Dissertation in zwei Hauptteile gegliedert. Der erste Teil dieser Arbeit zeigt wie sich ein Benutzer sicher mit einem Passwort authentifizieren kann, ohne dass dieses Benutzergeheimnis unnötigen Risiken ausgesetzt wird. Der zweite Teil erweitert digitale Signaturen um einige der Ideen, die im ersten Teil vorgestellt wurden.

Der erste Teil ist in drei Unterteile aufgespalten. Der erste Unterteil adressiert "Single Sign-On"-Protokolle (SSO), da existierende Protokolle einige gravierende Nachteile besitzen. Beispielsweise kann ein korrupter Server in Kerberos sich gegenüber Services als ein beliebiger Benutzer ausgeben und Benutzerpasswörter mit Wörterbuchattacken angreifen. Um diese Situation zu verbessern, werden zwei verteile SSO-Protokolle vorgestellt, in denen die Tokenerzeugung und die Passwortüberprüfung auf mehrere Server verteilt werden. Die dazugehörigen Funktionalitäten werden im UC-Framework definiert, welche zahlreiche handfeste Vorteile bietet. Beispielsweise wird die Sicherheit der realisierenden Protokolle in arbiträren Kontexten garantiert, während gleichzeitig einige praktische Limitierungen wie, zum Beispiel, Typos, korrelierende Passwortversuche und zufällig von einem Angreifer geratene Passwörtern direkt in der Definition verankert sind. Das erste realisierende Protokoll implementiert die Basisfunktionalität, während das zweite Protokoll erweitere Privatheitseigenschaften offeriert. Insbesondere lernen die Services im zweiten Protokoll nicht, welche anderen Zugriffberechtigungen ein Token hat, wie lange ein Token valide ist und Benutzer können ihre Identität hinter Pseudonymen verstecken.

Der zweite Unterteil stellt passwort-authentifizierte Signaturen vor, die virtuelle Smart-Cards realisieren, da reale Smart-Cards einige gravierende Nachteile besitzen. Beispielsweise sind immer spezielle Lesegeräte vonnöten, welche nicht immer zur Verfügung stehen, während es unrealistisch erscheint einem Benutzer zuzumuten diese immer bei sich führen. Virtuelle Smart-Cards umgehen diese Probleme indem ein persönliches Gerät, wie zum Beispiel ein Smart-Phone oder Tablet, benutzt werden kann um, nach einer zusätzlichen Passworteingabe, Signaturen auf beliebigen Nachrichten zu erzeugen. Diese Signaturen werden mit Hilfe einer zusätzlichen Servers generiert, um zu verhindern, dass im Falle einer Korruption des persönlichen Gerätes der private Signaturschlüssel dem Angreifer in die Hände fällt. Dieser Server hilft allerdings nur bei der Signaturerstellung, wenn das eingegebene Passwort korrekt war. Das vewendete Passwort

(und die Passwortversuche) sind nur angreifbar, wenn beide Entitäten korrumpiert wurden, während der Server die zu signierenden Nachrichten nicht lernt. Die dazugehörige Funktionalität ist, wie schon bereits für SSO, im UC-Framework definiert, also die gleichen Vorteile offeriert. Darüber hinaus ist das realisierende Protokoll sicher gegen adaptive Angreifer in einem neuen Korruptionsmodell. In diesem lernt der Simulator nicht alle vorherigen Ein- und Ausgaben der korrumpierten Partei. Dies modelliert, dass ein verlorenes Gerät dem Angreifer nicht alle verwendeten Passwörter aushändigt. Um das Protokoll beweisbar sicher zu machen, wird in diesem Unterteil ein neues Verschlüsselungsschema vorgestellt, welche es erlaubt Chiffrate für den Empfänger zu simulieren, allerdings sicheres Löschen voraussetzt. Die Implementierung zeigt, dass das Protokoll sogar schneller als moderne reale Smart-Cards ist.

Der dritte Unterteil stellt ein vollständig simulierbares Veschlüsselungsschema vor. Das im vorherigen Unterteil eingeführte Verschlüsselungsschema benötigt sicheres Löschen, was allerdings nicht immer zuverlässig realisiert werden kann. Um dieses Problem zu lösen, stellt dieser Unterteil eine erweiterte, und nicht-interaktive, Verschlüsselungsdefinition vor, welche es einem Angreifer erlaubt adaptiv Chiffrate erstellen zu lassen und, gleichzeitig, den verwendeten Zufall für die Schlüsselerzeugung und Chiffraterzeugung zu bekommen. Eine nicht-interaktive Definition ist insbesondere dann vonnöten, wenn Chriffrate weiterverwendet werden. Das wird dadurch demonstriert, indem die erste UC-Definition von adaptiv-sicherer Signcryption vorgestellt wird. Darüber hinaus wird auch gezeigt, dass weder solch ein Veschlüsselungsschema noch Signcryption in nicht-idealisierten Modellen realisierbar ist.

Der zweite Hauptteil dieser Arbeit behandelt digitale Signaturen und wie diese um zusätzliche Möglichkeiten erweitert werden können. Der erste Teil diskutiert schwärzbare Signaturen. In existierenden Definitionen von schwärzbaren Signaturen kann eine, teilweise vertrauenswürdige, dritte Partei (der "Sanitizer") Teile von signierten Nachrichten abändern. Allerdings ist für jeden ersichtlich welche Teile einer signierten Nachricht änderbar sind. Die neue Definition von unsichtbar schwärzbaren Signaturen fügt die Privatheitseigenschaft hinzu, dass nicht ersichtlich ist, welche Teile schwärzbar sind. Um diese Signaturen zu realisieren, werden Chameleon-Hashfunktionen mit flüchtigen Falltüren eingeführt. Diese Art von Hashfunktion erlaubt es beliebe Kollisionen zu finden, wenn zwei Geheimnisse zur gleichen Zeit bekannt sind. Eines dieser Geheimnisse ist ein geheimer Langzeitschlüssel, während der Zweite erst bei der Hashgenerierung erzeugt wird.

Der letzte Teil dieser Arbeit diskutiert das Problem wie Signaturerzeugungsrechte effizient entzogen werden können. Heutzutage muss bei jeder Signaturverifikation der Status des dazugehörigen öffentlichen Schlüssels überprüft werden. Dies benötigt im Regelfall Netzwerkzugriff und wirkt sich negativ auf die Effizienz aus. Die in diesem Unterteil vorgestellten Protokolle umgehen dies, indem sie die Zertifizierungsstelle schon bei Signaturerstellung beweisen lässt, dass der Schlüssel gültig ist. Um den gestiegenen Privatheitsanforderungen gerecht zu werden, lernt diese Entität nicht, welche Nachrichten signiert werden, während eine Erweiterung es sogar verhindert, dass Signaturen mit einem vorherigen Signaturerstellungsprotokoll verknüpft werden können.

Zusammengefasst stellt diese Dissertation beweisbar sichere, und effiziente, Protokolle vor, welche Alltagsprobleme lösen.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

Cryptography has penetrated the every-day life of virtually every person. For example, digital signatures, MACs, but also symmetric and asymmetric encryption are used on a regular basis on the Internet, e.g., every time a website is requested. Additional applications of cryptography include securing emails (e.g., PGP [Gar95]), attribute-based credentials (e.g., Idemix [CH02]), but also secure communication with governments (e.g., tax statements via the German ID-card [Dag13]). However, there are still a plethora of open real-world use-cases where cryptography can reduce the amount of work users have to perform, can improve a user's privacy or add additional possibilities, but have not yet been solved.

This thesis identifies a number of these challenges and provides solutions for them. Namely, the proposed constructions and primitives are of direct practical relevance, are geared to be highly efficient (way beyond "being in PPT" [End16]), are based on widely believed assumptions which stood the "test-of-time", while also providing state-of-the-art security and privacy. Moreover, this thesis takes care that the proposed primitives can easily be integrated into *existing* infrastructures to allow for an easier transition, i.e., for an easier deployment in today's environment. Thus, users directly benefit from the proposed ideas and do not require to change their way of thinking significantly, i.e., the primitives are useable as presented. Last, but not least, the introduced underlying primitives (see the next section) are designed to be easily re-useable in other contexts as new building blocks.

Roadmap. The rest of this chapter is structured as follows. An overview of how the author intends the reader to follow this thesis is given Section 1.1. Section 1.2 lists the concrete contributions of this thesis. The broader context of this thesis is defined in Section 1.3, followed by an introduction into the methodology used in Section 1.4.

1.1 How to Read This Thesis

This thesis consists of multiple chapters, which partially build on each other. Figure 1.1 depicts a flowchart depicting how the author intends the reader to follow the chapters. Namely, Chapter 1 (this chapter) is meant as the starting point for the reader, whereas Chapter 2 contains the preliminaries required to understand the definitions and primitives introduced in this thesis. Then, the thesis splits in two major parts. Chapter 3 to Chapter 5 form the first part, while

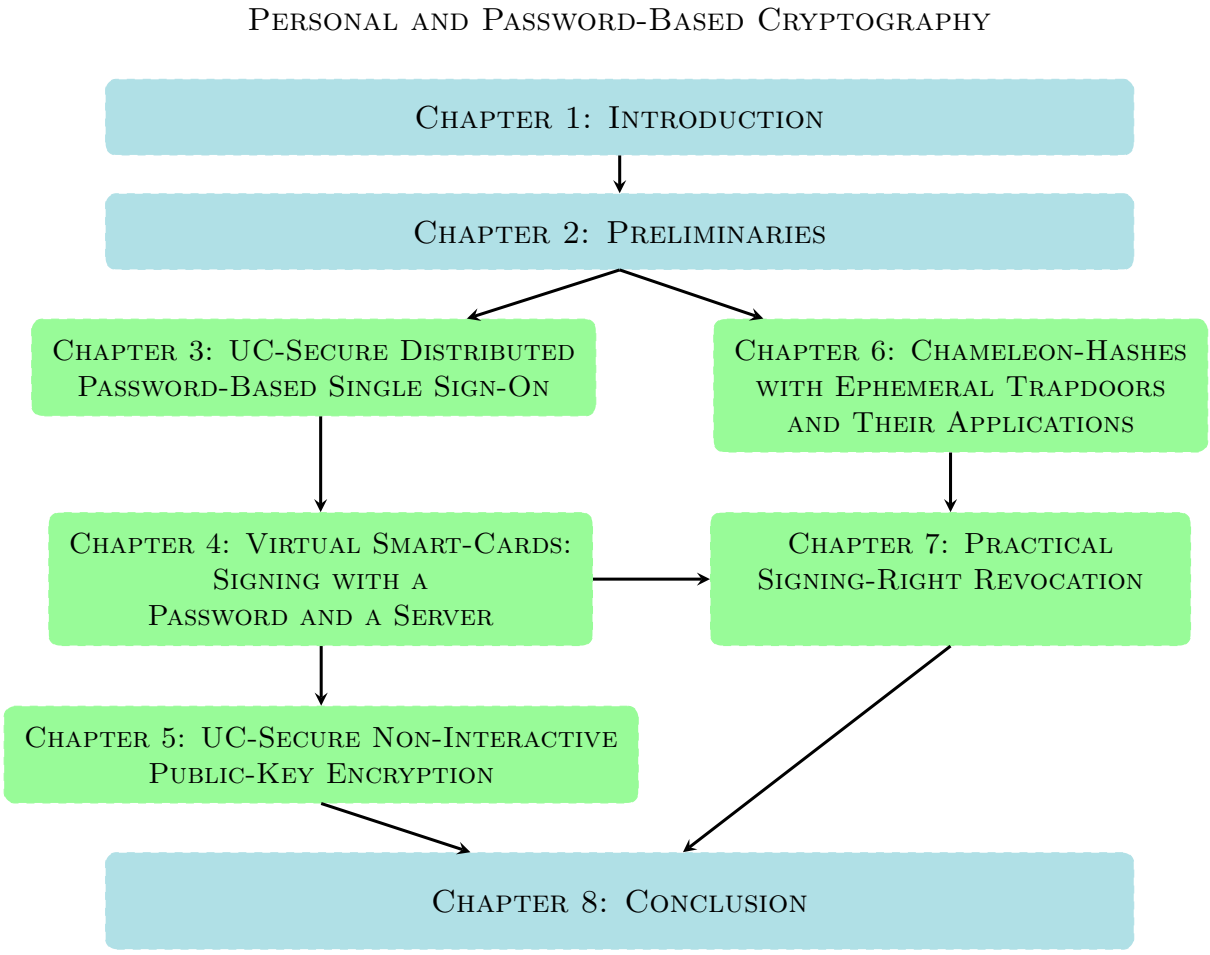

Figure 1.1: A flowchart describing how the ideas of the chapters are connected and are meant to be read. Boxes with a blue background do not contain any new major insights, while the ones with a green background do. The chapters on the left-hand side form the first part of thesis, while the right-hand side from the second part.

Chapter 6 and Chapter 7 shape the second part.

In the first part (Chapters 3-5), this thesis deals with the question of how users can use passwords to authenticate securely, i.e., how passwords can be used in a responsible way. Namely, passwords still are one of the most used and convenient means of user authentication, but usually do not contain enough entropy to be considered secure in all contexts. This also includes new building blocks, new corruption models and new insides how to model certain protocols.

The second part of this thesis (Chapter 6 and Chapter 7) deals with authentication of messages using signatures, also introducing new building blocks which can be used in other contexts as well, based on the ideas given in the first part of this thesis.

In more detail, Chapter 3 introduces password-based distributed UC-secure single sign-on, where a user has to authenticate with a password and its username and can later access a plethora of services without the need to authenticate again. Chapter 4 introduces the idea of password-authenticated server-assisted signing, including a new definition of non-committing encryption for the receiver. The underlying construction of that encryption scheme is then

shown to achieve UC-security even without secure erasures in Chapter 5. The second part of this thesis starts with the definition of chameleon-hashes with ephemeral trapdoors in Chapter 6, with the application of invisible sanitizable signatures. Then, Chapter 7 shows how to revoke signing rights without having the verifier query the revocation authority at each verification. Finally, this thesis is concluded in Chapter 8.

1.2 Contributions

As already outlined, the goal of this thesis is to propose practical (in the practitioner's sense) protocols and primitives, which are easily deployable in different contexts, while solving real-life problems, focusing on cases where authentication is a necessity.

In more detail, this thesis contains the following contributions, split into several chapters, partially building on each other.

Chapter 3 (UC-Secure Distributed Password-Based Single Sign-On). This chapter introduces distributed password-based UC-secure single sign-on (SSO). The idea of SSO is that a user only has to enter a password and can then use a plethora of services without re-entering the password every single time a service is accessed. However, in existing work a corrupt ticket-granting server can either impersonate users at services, can offline attack a user passwords and attempts as, e.g., possible in Kerberos, or does not model the case where an adversary was able to guess a password. To tackle this situation, this chapter introduces two provably secure protocols where the password check and token generation is distributed among multiple ticket-granting servers. Distributing the password check and token generation enforces that the adversary, as long as not a certain threshold of the servers granting the authentication tokens are corrupt, does not learn the password used for registration and, under no circumstances, anything about mistyped password attempts. The definition of SSO is given in the universal-composition (UC) framework. This guarantees security in arbitrary contexts, while also accounting for unavoidable practical implications such as typos, re-used passwords and the like into the definition by letting the environment choose these values. The environment also decides which services, and how long, a given participant can access for each generated token. Two protocols realizing SSO are presented. The first protocol, modeled as $\mathcal{F}_{\mathsf{sso}}^{1}$, offers the basic functionality of SSO, while the second extended protocol, given as $\mathcal{F}_{\mathsf{sso}}^{2}$, provides even more privacy guarantees. For example, in $\mathcal{F}_{\mathsf{sso}}^{2}$, the services do not learn which other access rights an entity has, how long a token is valid and allows to establish different identities with each service provider. Both protocols have been implemented. The corresponding evaluation shows that both protocols must be considered efficient enough for use in practice, especially considering their merits.

Chapter 4 (Virtual Smart-Cards). This chapter proposes the idea of virtual smart-cards, mitigating some problems of real smart-cards. For example, to use real smart-cards, specialized hardware readers are needed, but not always available, while carrying additional hardware is not very convenient for users. Moreover, attacks on hardware only become more and more powerful and puts the secret signing-key stored on the smart-card at high risk. Virtual smart-cards circumvent these problems by letting a user enter a, potentially low entropy, password on an

almost always available personal device, such as a smart-phone or tablet, to generate signatures on arbitrary messages with the help of an additional server. The main advantage of this approach is that the adversary needs to corrupt both the server and the personal device before gaining access to the signing-key, while the adversary also needs to correctly needs to guess a password before generating signatures becomes possible, even though the personal device was corrupted. The protocol is modeled as a UC-functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$, incorporating the same advantages as for the SSO protocols. The corresponding protocol is secure against adaptive corruptions in a new corruption model if the RSA assumption holds, assuming secure erasures. In more detail, in the new corruption model, the simulator no longer receives all prior input or output of the functionality at corruption. This models the case that the adversary does not learn the passwords entered on the device even if the personal device was corrupted at a later point in time. Thus, the adversary neither learns the password used at registration nor the password attempts, if at least either the device or the server remain honest. The chapter also introduces a new non-committing encryption definition, where the receiver can be corrupted at any time, while the adversary does neither receive the *randomness* used for ciphertext generation nor for secret key generation. The protocol has been implemented and even outperforms real smart-cards, demonstrated by the corresponding evaluation.

Chapter 5 (UC-Secure Non-Interactive Public-Key Encryption). The encryption scheme presented in the prior chapter inherently requires secure erasures. However, there are many situations where one does not want to, or cannot, assume secure erasures. This is, for example, the case with SSDs, where blocks are not really deleted, even if marked so. To tackle this, this chapter introduces a new non-committing encryption definition, where the adversary is allowed to receive *all* randomness used for secret key generation *and* encryption in a completely adaptive manner. This new definition is proven to be either stronger than, or incomparable to, widely used definitions which model security against adversaries which receive randomness. It is also proven to be equivalent to a completely local, i.e., non-interactive, definition of UC-secure encryption, coined $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, where the environment explicitly gains access to ciphertexts for further processing. The corresponding construction is derived from the encryption scheme introduced in Chapter 4. It is secure if trapdoor permutations exist in the programmable, and observable, random-oracle model. This definition is in particular useful if ciphertexts are processed further, e.g., for signing. This is shown by providing the first UC-secure signcryption definition $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ and construction secure against adaptive corruptions without secure erasures. This chapter also comes with an impossibility result: it is proven that any construction achieving this strong security notion cannot be realized in the standard model, i.e., not even a CRS helps. This impossibility result also extends to the signcryption functionality $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$.

Chapter 6 (Chameleon-Hashes with Ephemeral Trapdoors). This chapter introduces the idea of chameleon-hashes with ephemeral trapdoors and their application to invisible sanitizable signature schemes. In existing definitions of sanitizable signature schemes, a semi-trusted third party, i.e., the sanitizer, can alter signer-chosen blocks of signed messages. However, a third party can derive which parts are actually admissible, which lessens the privacy guarantees given. The new notion of invisible sanitizable signature schemes prohibits this leakage by additionally hiding the information which parts of a signed message are actually admissible, thus closing this gap. To build invisible sanitizable signature schemes, the new primitive of

chameleon-hashes with ephemeral trapdoors is introduced. Standard chameleon-hashes allow to find arbitrary collisions in the domain of a hash, if a trapdoor is known. This is sometimes too limiting, e.g., if finding collisions need to be prohibited. The new notion allows that the entity hashing towards a particular public key can prohibit finding collisions, as long as no second, i.e., an ephemeral, trapdoor is provided, which is generated during hash generation. Four constructions are given, based on RSA and the discrete-logarithm assumption in known-order groups. The first one is bootstrapped from a new "standard" chameleon-hash, while the second one is based on RSA-like assumptions in the random-oracle model. The last two constructions are based on the discrete-logarithm problem in known-order groups.

Chapter 7 (Practical Signing-Right Revocation). This chapter introduces a method to easily revoke signing rights. Existing solutions either require that a given certificate is checked whether it is revoked at every signature verification, requiring network access, or deploy other highly specialized, and mostly interactive, methods. This negatively impacts on the usefulness of this mechanism. This can be avoided, if the revocation status is already checked at signature generation by the signer, i.e., if the signature itself vouches for the fact that the corresponding public key was not revoked at signature generation time. Namely, the protocols proposed solve most of these problems by requiring that a semi-trusted third entity helps generating the signature if, and only if, the public key in question is not revoked. This semi-trusted third party does this without learning the message and, with an extension, cannot link signatures to protocols runs. The constructions are based on standard signatures, commitments and partially-blind signatures. This chapter also proposes some extensions which provide even more privacy, such as signer-anonymity, while also adding timestamps.

Chapter 8 (Concluding Remarks). This chapter summarizes this thesis and contains some concluding remarks concerning the contributions. This also includes possible future research directions which build upon the work presented.

1.3 Context of the Thesis

Here, the broad context of this thesis is defined, i.e., where this thesis is placed within the huge body of work already done in similar, and related, areas. In particular, as this thesis mostly deals with the authentication of humans and messages, there are two main contexts the thesis is embedded in. These are presented next.

1.3.1 Human Authentication

Clearly, authenticating humans is one of the most basic tasks carried out. Therefore, there are many general approaches. This distinction is mostly relevant for the first part of this thesis.

In a nutshell, one can distinguish between the following ideas [BJR+06], which summarize the currently most deployed solutions:

1.3.1.1 What you are

This is normally related to *biometrics*, e.g., someone's voice [KAKP06], DNA [JRP04], blood vessel patterns [FRH+12] and so on. One of the advantages of this method is that one normally cannot lose, or forget, what and who someone is, ignoring accidents and the like. However, most of the hardware used to check these biometric characteristics are expensive, cumbersome to use, are not available everywhere, while the data collected needs to be checked against the data in the database, which is clearly privacy invading, if one does not take extra precautions [RCB01].

1.3.1.2 What you have

What you have is defined as hardware and the stored information which the person wanting to authenticate owns. This may be smart-cards [CJT02], certain tokens [SS96], but also everyday hardware such as car keys [dKGHG08], passports [KR00a] and the like. Even though hardware can store cryptographically secure keys, they have some serious drawbacks. In particular, they can be lost and with a bit of effort, such keys may even be extracted, cloned or otherwise be manipulated [KK99, VWG07]. Same of these drawbacks can be tackled by "physically uncloneable functions" (PUF), but here a loss, or defect, implies a complete loss of the data stored [BFSK11, SBP16, XSA+16]. This is clearly not acceptable in all cases.

1.3.1.3 What you know

What you know is data one remembers, e.g., a password, pass-phrase, memories [NP97] and so on. Normally, this data does not contain very much entropy, can be forgotten and easily be guessed [Gos12]. On the upside, designing a system around this paradigm is way easier than using the other ones, as, e.g., for passwords only a standard keyboard is required. However, it is well-known that relying on knowledge is rather dangerous, e.g., due to social engineering [WD95, Wor08] or malware on devices [RAB14]. Moreover, as passwords and the like normally follow some non-quantifiable distribution, i.e., if the distribution is not explicitly pre-defined [HSBB16, MKV+13, MUS+16, USB+15], it is very dangerous to argue about security, if the passwords are used as proper cryptographic material. For example, some work related to password-based authentication comes without UC-security [Kie16], where security is defined over some dictionary. However, it is not exactly clear what security guarantees are given in this case, as the success probability may not be negligible or the passwords are drawn from artificial distributions [CHK+05b]. Moreover, passwords are normally used in a way which is plainly irresponsible, e.g., simple salting [FH07], which is essentially useless considering modern computing power [Gos12].

The case "somebody you know", introduced by Brainard et al. [BJR+06], is ignored here, as it seems to be more of relevance in the context of social networks. However, it may lead to additional insights. Clearly, the above examples are far away from being exhaustive. However, they allow a first distinction of the methods used.

1.3.1.4 Relevance for the Methodology

It is very easy to combine two or more methods to counter some of the problems discussed before [Sch05]. This is called "multi-factor authentication". The first part of this thesis is

located within the cases "what you know" and "what you own". Namely, proper cryptographic keys are defined to be related to the case "what you own", while passwords are part of "what you know". The first part of this thesis bridges the gap between "what you own" and "what you know", while also presenting some additional new primitives which can be used in this and others contexts as well.

Summarized, the idea is to provide secure password-based primitives in an environment where normally proper cryptographic keys, i.e., high-entropy values, are required, but cannot be stored or handled and thus are not available.

1.3.2 Message Authentication

The second part (and partially Chapter 4 in the first part of this thesis), deals with signatures, which authenticate messages. In a traditional sense, signatures allow a holder of secret key to sign a message, while the resulting signature can be verified by every party holding the corresponding public key pk [DH76, Gam84, GMR88, Mer87, RSA78]. Clearly, only the holder of the secret key should be able to sign messages. However, it became clear very quickly that this may not be enough and therefore a lot of extensions have been proposed. For example, some signature schemes allow for efficient zero-knowledge proofs [AFG+10, CL02b, LMPY16], which can, e.g., be used in the context of anonymous digital credentials [CH02]. Related to this approach is the idea of having a set of participants collaborating, e.g., threshold signatures [Boy89], proxy signatures [BPW12b, MUO96], group signatures [BCK+14, BCN+10, CvH91] and attribute-based credentials [MPR11]. Here, it is still clear that the entity (or a set of entities) creating the signature is still in charge of the message itself. However, these signatures still prevent that a third party can generate new signatures on derived data, still validating under the original public keys. Obviously, such a feature has its merits as well. For example, it may be useful to derive means from a set of signed data, disclose only a certain subset of signed data to third party (which is actually a feature in most anonymous credential systems [CH02]) or change certain parts of signed messages, e.g., as an additional solution to the long-standing DNS enumeration problem [GNP+15].

1.3.2.1 Computing on Authenticated Data

Following this line of ideas, it was only a matter of time till schemes were proposed which allow to alter messages in a controlled way. Here, "altering" means that a second entity, perhaps not being the original signer, can compute certain functions on signed data to derive a signature which verifies under the original signer's public key. What functions can be computed depends on the scheme in question. For example, redactable signature schemes [JMSW02, KB13, PS14, SR10, SBZ01] allow to remove parts of signed data, while append-only signatures only allow to add new messages to a signature [KMPR05]. Somewhere in between are sanitizable signatures [ACdMT05, BFF+09], which allow to change signer-chosen parts of a message to be altered to arbitrary bit-strings. These approaches have then been generalized [ABC+15, ALP12, BMS16]. Another line of work proposes that someone can calculate functions on signed data, such as means, standard derivations and so on [BF11b, BGI14, LPJY15].

1.3.2.2 Revocation of Signatures and Signing-Rights

One problem of signatures, which is very often overlooked, is how to handle the case where a secret key is lost, i.e., given to the adversary [BS01b]. In particular, what happens to the still honestly generated signatures? Can the adversary still generate validating signatures? In the context of anonymous credentials, this is easily solved by revoking the right to generate authentication tokens (which are essentially zero-knowledge proofs [QQQ+89] on signatures), which only have a limited life-span anyway [CDR16, CKL+15, CKS09, CKS10]. In the context of standard, i.e., "long-term" signatures, however, the problem is far more severe. In particular, if one wants to revoke a public key, all signatures become invalid, even the ones generated by the legitimate holder of the secret key. This may not always be wanted.

1.3.2.3 Relevance for the Methodology

The second part of this thesis deals with signatures in the general sense. It introduces a new type of sanitizable signature scheme with strengthened privacy definitions, whereas the underlying new building block can be re-used in different contexts as well. The last chapter introduces an efficient protocol which partially solves the problem how to revoke signing rights without revoking all signatures at once, while also circumventing the problem that verification needs up-to-date revocation lists.

Additional related work is presented in the respective chapters, as state-of-the-art is better understood in context with the respective contributions.

1.4 Provable Security in a Nutshell

This thesis exclusively deals with the topic of provable security.[1] This means that the security of the primitives presented is mostly based on problems from complexity theory which are believed to be hard, dating back to the ideas by Diffie and Hellman [DH76].[2] This is in contrast to "old" approaches, where one simply assumes the security of a primitive. Simply assuming security normally leads to broken schemes, e.g., the ciphers used during the second world war or Caesar's cipher. Thus, the paradigm shift to base security on, e.g., complexity-based problems became a necessity. Even though this seems to be a very limiting constraint, there are a plethora of assumptions which can be used, e.g., computing e^{th} roots in a group of unknown order [RSA78], discrete logarithms (and variants thereof) [DH76], learning parity with noise (LPN) [BKW03, Pie12], learning with errors (LWE) [Reg09], but also some more "esoteric" assumptions such as the "Uber-Assumption" [Boy08]. Some of them are even assumed to be hard to solve using (currently highly hypothetical) quantum-computers [Reg04]. In more detail, provable security means that if there is a successful attacker $\mathcal{A}$ on a cryptographic scheme $\mathcal{S}$, then this attacker $\mathcal{A}$ can be used to break a certain problem $\mathcal{P}$ (sometimes there is a set of problems). In turn, this means that the hardness of the problem $\mathcal{P}$ implies the security of the cryptographic scheme $\mathcal{S}$. In other words, the security of $\mathcal{S}$ can be *reduced* to the hardness of $\mathcal{P}$. To transform such an adversary $\mathcal{A}$, a PPT reduction $\mathcal{R}$ (sometimes called the adversary $\mathcal{B}$) is constructed which, together with $\mathcal{A}$, breaks $\mathcal{P}$ with a probability polynomially related to

[1] Not to be confused with "probably secure".

[2] Mostly, in this context, means that one can also resort to, e.g., statistical arguments.

the probability that $\mathcal{A}$ breaks $\mathcal{S}$. To be meaningful, the reduction $\mathcal{B}$ is also required to run in polynomial time. Thus, an ideal reduction $\mathcal{B}$ has roughly the same probability and running time as the adversary $\mathcal{A}$, i.e., it is "tight". A "low-quality" reduction requires much more computation, i.e., time, or is very unlikely to break the problem $\mathcal{P}$, even though the adversary $\mathcal{A}$ is successful. This gap is also referred to as "loss". Bader et al. provide a very good overview [BJLS16]. In this thesis, two different methods to define provably secure schemes are used, which are introduced in a more formal manner next.

1.4.1 Game-Based Security

A game-based definition consists of two entities [BR06, Sho04]. Namely, there is a challenger $\mathcal{C}$ and the corresponding adversary $\mathcal{A}$. The underlying idea is that the adversary $\mathcal{A}$ "plays a game" with the challenger $\mathcal{C}$. In particular, the challenger defines the environment of the adversary such as oracles, but also some precisely defined winning conditions and the inputs to the adversary. These winning conditions can, e.g., be outputting a signature σ^* on some message m^* verifying for some honestly generated (thus generated by the challenger $\mathcal{C}$) public key pk for which the adversary $\mathcal{A}$ has never seen a signature. Thus, the adversary $\mathcal{A}$ is said to "win the game", if it breaks the scheme $\mathcal{S}$ in question.

Normally, game-based definitions are used for more basic primitives such as signature schemes, encryption schemes and so forth. However, game-based definitions are very delicate, are hard to write for more complex primitives, and also do not define correctness of the scheme in question. This becomes even harder, if one wants to define protocols. To tackle this, one can use simulation-based security definitions, introduced next.

1.4.2 Simulation-Based Security

Simulation-based security dates back to the ideas given by Goldreich et al. [GMW87]. Sometimes, this is also called "ideal/real world paradigm". In a nutshell, the adversary is no longer required to break a primitive, but only to distinguish between an "ideal world", which may not even contain any cryptographic details, yet defines the input-output behavior in form of an ideal functionality which is perfectly correct (thus, defining correctness on-the-fly) and perfectly secure by definition, with the "real world", where the cryptographic protocol is executed [Can00]. The idea is that if there exists a simulator SIM, which only receives the information explicitly leaked from the ideal functionality, yet can produce a view which is indistinguishable from the real protocol execution, an adversary can run this simulator SIM itself and thus does not learn any additional information from the real execution. This paradigm is in particular suitable for more complex protocols, as the ideal functionality encapsulates "the essence" of the protocol, including what an adversary learns. This will become clearer, when some ideal functionalities, as well as the corresponding protocols and proofs, are presented in Chapter 2. As before, the idea is that once such a distinguisher exists, some hard problem $\mathcal{P}$ can be solved using this distinguisher. Moreover, if one resorts to an even more restricted simulation-based framework such as UC [Can01], one can even achieve security with concurrent executions in arbitrary contexts.

For both paradigms, this thesis only deals with black-box access to the adversary, as well as reductions and simulators which are assumed to be (non-)uniform polynomially bounded, which

is in contrast to super-polynomial simulations [BS05].

A small introduction is presented in Chapter 2. If more details, and formal definitions, are required, the reader is referred to the work done by Baecher et al. [BBF13], as well as Bernhard and Warinschi [BW13], Brzuska's thesis [Brz13] and the tutorial by Lindell [Lin17].

1.5 A Personal Note

The thesis is (hopefully) written in such a way that a reader with a decent comprehension of cryptography is able to understand the ideas given - and with a bit of more work - even the proofs. In other words, the thesis wants to be readable, even for non-expert readers. It was also taken care that this thesis is complete, meaning that all preliminaries are given, or if too basic, cited. Thus, the author's intention is to provide a really practical thesis where everyone, with a bit of effort, can follow the ideas and bring them into practice. The author is, and will be (at least if possible ⌣̈), available for any questions regarding the results.

Chapter 2

Preliminaries

Now, the basic preliminaries used throughout this thesis are introduced. These definitions can be read on a "as-needed-basis", as most of them are standard. Less well-known definitions are introduced when used in the respective chapters.

Roadmap. The roadmap of this chapter is as follows. Section 2.1 introduces the used notation. The used cryptographic hardness assumptions used are presented in Section 2.2, while Section 2.3 introduces some already known cryptographic primitives such as digital signature schemes and the like. Section 2.4 provides a short introduction of the universal composition (UC) framework, including some standard functionalities, such as the authenticated channel functionality $\mathcal{F}_{\mathsf{Auth}}$, while Section 2.5 introduces the random-oracle methodology.

2.1 Notation and Conventions

With $\lambda \in \mathbb{N}$ the main security parameter is denoted. Likewise, 1^λ is the string of λ ones. All algorithms, and the adversary $\mathcal{A}$, receive 1^λ as an, often implicit, input. An adversary $\mathcal{A}$ with binary output is called a distinguisher D. If a is assigned a random element chosen uniformly from a set S, $a \xleftarrow{\$} S$ is used. If A is a PPT (efficient) algorithm $y \xleftarrow{\$} A(x;r)$ is used to denote that y is assigned the output of A with input x and external random coins r. If r is dropped, the random coins are freshly drawn internally. The notation $(y;r) \xleftarrow{\$} A(x)$ means that r is assigned the value of the fresh random coins used by A. The randomness space is denoted as $\mathcal{R}$ and may depend on some external parameters such as a public key. For deterministic algorithms, $y \leftarrow A(x)$ is used. The notation $\mathcal{A}^{\mathcal{O}^{\mathsf{function}}(\cdot,\cdot,b)}$ means that the adversary $\mathcal{A}$ has oracle access to a function $\mathsf{function}$ which requires two additional parameters chosen by the adversary $\mathcal{A}$, while b is fixed. All algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. This is not always made explicit for the sake of readability. A function $\nu : \mathbb{N} \to [0,1]$ is negligible if $\nu(\lambda) \in \lambda^{-\omega(1)}$. The bit-wise exclusive OR (XOR) is denoted as $\oplus : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$. Note, XOR is an involution. The message space of a primitive is the set $\mathcal{MS}$. As for the randomness space $\mathcal{R}$, $\mathcal{MS}$ may implicitly depend on system parameters or a public key. For simplicity, it is assumed that $\mathcal{MS} = \{0,1\}^*$, where $\{0,1\}^*$ is the set of all finite strings with some length polynomial in the security parameter λ if not explicitly defined otherwise. To further shorten notation, $\widetilde{X}_{n,m}$, where $n \leq m$, denotes the vector $(X_n, X_{n+1}, X_{n+2}, \ldots, X_{m-1}, X_m)$. The

notation $[1, n]$ is a shorthand notation for the set $\{1, 2, \ldots, n\}$. By $|m|$, where m is a binary string, the binary length is denoted. The notation $|S|$ denotes the cardinality of a set S. If an argument is a list, an injective encoding which allows embedding it into $\{0,1\}^*$ is required. The function $\varphi : \mathbb{N} \to \mathbb{N}$ denotes Euler's totient function. Further, it is sometimes (implicitly) required that a public key pk can always be uniquely derived from the corresponding secret key sk. This can simply be achieved by appending the randomness used to create sk to sk itself. This is not be made explicit to avoid unhelpful boilerplate notation. For certain security properties it is required that values only have one canonical representation, e.g., a "4" is not the same as a "04", even if seen as an element of $\mathbb{N}$. The notation $\mathbb{G}^\times$ for $\mathbb{G} \setminus \{1_\mathbb{G}\}$ is used, where $\mathbb{G}$ is some multiplicatively written group and $1_\mathbb{G}$ the corresponding trivial normal subgroup.[1] All mathematical symbols, such as $\mathbb{Z}$, $\vee$, $\cup$ etc., have their usual meaning, if not explicitly defined otherwise. Sets of parties are denoted as S, while single parties are denoted as $\mathcal{S}$.

2.2 Cryptographic Hardness Assumptions

As this thesis deals with the topic of *provable* security, some assumptions are needed on which the security of the proposed schemes are based on. Throughout this thesis, only assumptions which "stood the test-of-time" are used to avoid less-studied ones, which have not been thoroughly analyzed. If existing assumptions are used only once, they are introduced when needed. This is done to keep this chapter short.

2.2.1 Computational Indistinguishability of Distributions

At certain locations it is required that certain distributions are computationally indistinguishable. More formally, consider the following definition taken from Katz and Lindell [KL07, p 232].

Definition 2.1 (Computational Indistinguishability)**.** *Two probability ensembles $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable, denoted as $X \approx_c Y$, if for every (non-uniform) PPT distinguisher D there exists a negligible function ν such that:*

$$|\Pr[D(X_\lambda) = 1] - \Pr[D(Y_\lambda) = 1]| \leq \nu(\lambda)$$

In other words, the distinguisher D cannot decide, except with negligible probability in λ, whether it sees a sample from the first or the second distribution, depending on λ. Sometimes, "indistinguishable" instead of "computationally indistinguishable" is used.

2.2.2 The Discrete-Logarithm Assumption

Let $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$ be a group generator for a prime-order, and multiplicatively written, group $\mathbb{G}^2$, along with a generator g of order q of that group, such that $\langle g\rangle = \mathbb{G}$. The discrete-logarithm (DL) problem associated to DLGen is defined as follows. Given $\mathbb{G}$, g, q, and g^x, where $x \xleftarrow{\$} \mathbb{Z}_q$, to find this x [KL07, p 278].

[1] Note, this is different from $\mathbb{G}^*$.

[2] Clearly, DLGen only returns a description of $\mathbb{G}$.

Definition 2.2 (Discrete-Logarithm Assumption)**.** *The discrete-logarithm assumption holds, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr[(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda), x \xleftarrow{\$} \mathbb{Z}_q, x' \xleftarrow{\$} \mathcal{A}(\mathbb{G}, g, q, g^x) : x = x'] \leq \nu(\lambda)$$

Sometimes samples from $\mathbb{Z}_q^*$ are drawn instead of $\mathbb{Z}_q$. This changes the view of the adversary only negligibly and is thus not made explicit.

2.2.3 The Decisional Diffie-Hellman Assumption

Let $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$ be a group generator for a prime-order, and multiplicatively written, group $\mathbb{G}$[3], along with a generator g of order q of that group, such that $\langle g \rangle = \mathbb{G}$. The decisional Diffie-Hellman (DDH) problem associated to DLGen is defined as follows. Given $\mathbb{G}$, g, q, g^x, g^y, g^z, decide whether $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q$ [KL07, p 279].

Definition 2.3 (Decisional Diffie-Hellman Assumption)**.** *The decisional Diffie-Hellman assumption holds, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\left| \Pr \left[\begin{array}{c} (\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \\ (x, y, z) \xleftarrow{\$} \mathbb{Z}_q^3, a \xleftarrow{\$} \mathcal{A}(\mathbb{G}, g, q, g^x, g^y g^{b(xy)+(1-b)z}) : a = b \end{array} \right] - \frac{1}{2} \right| \leq \nu(\lambda)$$

Again, sometimes samples from $\mathbb{Z}_q^*$ are drawn instead of $\mathbb{Z}_q$. This changes the view of the adversary only negligibly and is thus not made explicit.

2.2.4 Bilinear Maps

For some constructions, bilinear maps are used. These are introduced next. In particular, let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic multiplicative groups with prime order q, generated by g_i, i.e. $\mathbb{G}_i = \langle g_i \rangle$ for $i \in \{1, 2, T\}$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map such that:

1. Bilinearity: $\forall u \in \mathbb{G}_1, \forall v \in \mathbb{G}_2 : \forall a, b \in \mathbb{Z}_q : e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $\exists u \in \mathbb{G}_1, \exists v \in \mathbb{G}_2 : e(u, v) \neq 1$, i.e., $e(g_1, g_2) = g_T$.
3. Computability: There is an efficient algorithm that calculates the mapping e.

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda)$ be a group generator for three prime-order, and multiplicatively written, groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$[4], along with the corresponding generators g_1, g_2, g_T of order q of each group, such that $\langle g_1 \rangle = \mathbb{G}_1$, $\langle g_2 \rangle = \mathbb{G}_2$, and $\langle g_T \rangle = \mathbb{G}_T = \langle e(g_1, g_2) \rangle$, where e is the bilinear map. The discrete-logarithm (DL) problem associated to BLGen is defined as follows. Given $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, g_1, g_2, g_T, q, e, and g_2^x, where $x \xleftarrow{\$} \mathbb{Z}_q$, to find this x.

Definition 2.4 (Discrete-Logarithm Assumption for $\mathbb{G}_i$)**.** *The discrete-logarithm assumption for $\mathbb{G}_i$ now states that for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr \left[\begin{array}{c} (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda), \\ x \leftarrow \mathbb{Z}_q, x' \xleftarrow{\$} \mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, g_i^x) : x = x' \end{array} \right] \leq \nu(\lambda)$$

[3] Clearly, DLGen only returns a description of $\mathbb{G}$.

[4] Clearly, BLGen only returns a description of the groups.

This clearly also implies that DL is hard in $\mathbb{G}_T$.

Sometimes samples from $\mathbb{Z}_q^*$ are drawn instead of $\mathbb{Z}_q$. This changes the view of the adversary only negligibly and is thus not made explicit. Moreover, it is also required that the pairing is of type III, i.e., there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ in any direction [GPS08].

2.2.5 The RSA Assumption

Let $(N, e, d, p, q) \xleftarrow{\$} \mathsf{RSAGen}(1^\lambda)$ be an RSA-key generator returning an RSA modulus $N = pq$, where p and q are random distinct primes, $e > 1$ an integer co-prime to $\varphi(N)$ and $d \equiv e^{-1} \bmod \varphi(N)$. The RSA one-wayness problem associated to RSAGen is, given N, e and $y \xleftarrow{\$} \mathbb{Z}_N^*$, to find x such that $x^e \equiv y \bmod N$ [RSA78].

Definition 2.5 (RSA Assumption)**.** *The RSA (one-wayness) assumption associated to* RSAGen *holds, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr[(N, e, d, p, q) \xleftarrow{\$} \mathsf{RSAGen}(1^\lambda), y \xleftarrow{\$} \mathbb{Z}_N^*, x \xleftarrow{\$} \mathcal{A}(N, e, y) : x^e \equiv y \bmod N] \leq \nu(\lambda)$$

2.2.6 The One-More RSA-Assumption

The one-more RSA assumption associated to RSAGen is provided an inversion oracle $\mathcal{I}$ which inverts any element $x \in \mathbb{Z}_N^*$ w.r.t. e and a challenge oracle $\mathcal{C}$, which, at each call, returns a random element $y_i \in \mathbb{Z}_N^*$, to, given N and e, invert more elements received by the challenge oracle than calls to the inversion oracle.

Definition 2.6 (One-More RSA Assumption)**.** *The one-more RSA assumption holds, if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[(N, p, q, e, d) \leftarrow \mathsf{RSAGen}(1^\lambda), X \leftarrow \mathcal{A}(N, e)^{\mathcal{C}(n), \mathcal{I}(d, n, \cdot)} : \\ \textit{more values returned by } \mathcal{C} \textit{ are inverted than queries to } \mathcal{I}] \leq \nu(\lambda)$$

Here, X is the set of inverted challenges [BNPS03].

For all the RSA assumptions, it is sometimes required that e is larger than any possible n w.r.t. λ (or even for $e > n^3$) and that it is prime. Re-stating the assumptions with these conditions is straightforward. In this case, it is also required that e is drawn independently from p, q, or n (and d is then calculated from e and not vice versa). This can, e.g., be achieved by requiring that e is drawn uniformly from $[n + 1, \ldots, 2n] \cap \{p \mid p \text{ is prime}\}$, where n is the largest RSA modulus possible w.r.t. to λ. This is left to the concrete instantiation of RSAGen.

2.3 Cryptographic Primitives

Now some basic cryptographic primitives are introduced, which are used throughout this thesis.

Experiment $\mathsf{PR}_{\mathcal{A}}^{\mathsf{PRG}}(\lambda)$
$b \xleftarrow{\$} \{0,1\}$
if $b = 0$, let $v' \xleftarrow{\$} \{0,1\}^{2\lambda}$
else, let $v \xleftarrow{\$} \{0,1\}^{\lambda}$ and $v' \leftarrow \mathsf{Eval}_{\mathsf{PRG}}(v)$
$a \leftarrow \mathcal{A}(v')$
return 1, if $a = b$
return 0

Figure 2.1: PRG Pseudo-Randomness

2.3.1 Collision-Resistant Hash-Functions

In a nutshell, a family $\{\mathcal{H}_{\mathcal{R}}^k\}_{k\in\mathcal{K}}$ of hash-functions $\mathcal{H}^k : \{0,1\}^* \to \mathcal{R}$ indexed by key $k \in \mathcal{K}$ is collision-resistant, if an adversary cannot find any non-trivial collisions [Rog06].

Definition 2.7 (Collision-Resistant)**.** *A family $\{\mathcal{H}_{\mathcal{R}}^k\}_{k\in\mathcal{K}}$ of hash-functions $\mathcal{H}^k : \{0,1\}^* \to \mathcal{R}$ indexed by key $k \in \mathcal{K}$ is collision-resistant, if for any PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr[k \xleftarrow{\$} \mathcal{K}, (v, v') \xleftarrow{\$} \mathcal{A}(k) : \mathcal{H}_{\mathcal{R}}^k(v) = \mathcal{H}_{\mathcal{R}}^k(v') \ \wedge \ v \neq v'] \leq \nu(\lambda)$$

2.3.2 Pseudo-Random Generators

PRGs allow to generate randomness from a short seed which is indistinguishable from a purely random string. For this thesis, a constant stretching factor of 2 is assumed, as this is enough to prove security. However, different stretching factors are also possible [App12].

Definition 2.8 (Pseudo-Random Generators)**.** *A pseudo-random generator* PRG *has one algorithm* $\{\mathsf{Eval}_{\mathsf{PRG}}\}$ *such that:*

$\mathsf{Eval}_{\mathsf{PRG}}$**.** *The deterministic algorithm* $\mathsf{Eval}_{\mathsf{PRG}}$ *gets as input a point $p \in \{0,1\}^{\lambda}$ to evaluate. It outputs a new point $p' \in \{0,1\}^{2\lambda}$:*

$$p' \leftarrow \mathsf{Eval}_{\mathsf{PRG}}(p)$$

Security. For security, it is required that PRG is actually pseudo-random [KL07, p 86].

Definition 2.9 (PRG Pseudo-Randomness)**.** *A pseudo-random generator* PRG *is called pseudo-random, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathsf{PR}_{\mathcal{A}}^{\mathsf{PRG}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.1.

Note, only one sample is provided to $\mathcal{A}$. Security for multiple samples follows from a simple hybrid-argument, i.e., a sequence of games [KL07, p 214].

Definition 2.10 (Secure PRGs)**.** *A pseudo-random generator* PRG *is secure, if it is pseudo-random.*

Experiment $\mathsf{PR}_{\mathcal{A}}^{\mathsf{PRF}}(\lambda)$
$\kappa \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{PRF}}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$f \xleftarrow{\$} F_\lambda$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Eval}'_{\mathsf{PRF}}(\kappa,\cdot)}}(1^\lambda)$
where oracle $\mathsf{Eval}'_{\mathsf{PRF}}$ on input κ and p:
return $\perp$, if $p \notin \{0,1\}^\lambda$
if $b = 0$, return $\mathsf{Eval}_{\mathsf{PRF}}(\kappa, p)$
return $f(p)$
return 1, if $a = b$
return 0

Figure 2.2: PRF Pseudo-Randomness

2.3.3 Pseudo-Random Functions

PRFs allow the holder of a, potentially secret, key κ to evaluate a function on an additional input point p.

Definition 2.11 (Pseudo-Random Functions)**.** *A standard pseudo-random function* PRF *consists of two algorithms* $\{\mathsf{KeyGen}_{\mathsf{PRF}}, \mathsf{Eval}_{\mathsf{PRF}}\}$ *such that:*

$\mathsf{KeyGen}_{\mathsf{PRF}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{PRF}}$ *outputs a function key* $\kappa \in \{0,1\}^\lambda$*, where* λ *is the security parameter:*

$$\kappa \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{PRF}}(1^\lambda)$$

$\mathsf{Eval}_{\mathsf{PRF}}$**.** *The deterministic algorithm* $\mathsf{Eval}_{\mathsf{PRF}}$ *gets as input the secret key* κ *and a point* $p \in \{0,1\}^\lambda$ *to evaluate. It outputs a new point* $p' \in \{0,1\}^\lambda$*:*

$$p' \leftarrow \mathsf{Eval}_{\mathsf{PRF}}(\kappa, p)$$

Sometimes, the input is not an element from $\{0,1\}^\lambda$, but some group-element $g \in \mathbb{G}$. The definitions in this case are similar.

Security. For security, it is required that PRF is actually pseudo-random [KL07, p 86].

Definition 2.12 (PRF-Pseudo-Random)**.** *A pseudo-random function* PRF *is pseudo-random, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{PR}_{\mathcal{A}}^{\mathsf{PRF}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.2, where $F_\lambda = \{f : \{0,1\}^\lambda \to \{0,1\}^\lambda\}$ *is the set of all functions* f *mapping a value* $v \in \{0,1\}^\lambda$ *to another value* $v' \in \{0,1\}^\lambda$*.*

Definition 2.13 (Secure PRFs)**.** *A pseudo-random function* PRF *is secure, if it is pseudo-random.*

2.3.4 One-Way Trapdoor Permutations

In a nutshell, the idea of TDPs is that a permutation can only be evaluated in one direction. The other direction can only be calculated if an additional secret, i.e., the trapdoor, is known.

Definition 2.14 (One-Way Trapdoor Permutations)**.** *A one-way trapdoor permutation* TDP *consists of four algorithms* $\{\mathsf{KeyGen}_{\mathsf{TDP}}, \mathsf{Eval}_{\mathsf{TDP}}, \mathsf{Inv}_{\mathsf{TDP}}, \mathsf{Sample}_{\mathsf{TDP}}\}$ *such that:*

$\mathsf{KeyGen}_{\mathsf{TDP}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{TDP}}$ *is an instance generator, generating a description* Σ *of some alphabet, a public key* f *and a private key* f^{-1}*, such that:*

$$(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$$

From now on, Σ *is treated as implicitly contained in* f*. Moreover, it is required that* $|\Sigma| \geq 2^{2\lambda}$*.*

$\mathsf{Eval}_{\mathsf{TDP}}$**.** *The deterministic algorithm* $\mathsf{Eval}_{\mathsf{TDP}}$ *gets as input a value* $s \in \Sigma$*, and the public key* f*. It outputs a new point* $s' \in \Sigma$*:*

$$s' \leftarrow \mathsf{Eval}_{\mathsf{TDP}}(f, s)$$

The notation $s' \leftarrow f(s)$ *is used to abbreviate* $s' \leftarrow \mathsf{Eval}_{\mathsf{TDP}}(f, s)$ *from now on.*

$\mathsf{Inv}_{\mathsf{TDP}}$**.** *The deterministic algorithm* $\mathsf{Inv}_{\mathsf{TDP}}$ *gets as input a value* $s \in \Sigma$*, and the private key* f^{-1}*. It outputs a new point* $s' \in \Sigma$*:*

$$s' \leftarrow \mathsf{Inv}_{\mathsf{TDP}}(s)$$

The notation $s' \leftarrow f^{-1}(s)$ *is used to abbreviate* $s' \leftarrow \mathsf{Inv}_{\mathsf{TDP}}(f^{-1}, s)$ *from now on.*

$\mathsf{Sample}_{\mathsf{TDP}}$**.** *The algorithm* $\mathsf{Sample}_{\mathsf{TDP}}$ *returns a uniformly distributed value* $s \in \Sigma$*:*

$$s \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(\Sigma)$$

For simplicity, the convention that the randomness space of $\mathsf{Sample}_{\mathsf{TDP}}$ *has the same size as* Σ *is used.*

Note, the randomness for sampling is explicitly modeled, as they are some cases where a distinction is of paramount importance [GR13].

Correctness. It is required that for all $\lambda \in \mathbb{N}$, all $(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$, f and f^{-1} both define a permutation over Σ. Thus, for all $s \in \Sigma$ it must hold that $s = f^{-1}(f(s))$ and $s = f(f^{-1}(s))$.

Security. For security, it is required that a TDP is actually one-way without knowing the trapdoor [KL07, p 93]. More formally, for all PPT adversaries $\mathcal{A}$ there exists a negligible function ν such that:

$$\Pr[(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda), x \xleftarrow{\$} \Sigma : x = \mathcal{A}(f, f(x), \Sigma)] \leq \nu(\lambda)$$

As an example, an RSA-key generator RSAGen yields a trapdoor one-way permutation under the RSA assumption with $\Sigma = \mathbb{Z}_N^*$, $f(x) := x^e \bmod N$ and $f^{-1}(y) := y^d \bmod N$ [GM84].

Definition 2.15 (Secure TDPs)**.** *A one-way trapdoor permutation* TDP *is secure, if it is correct and one-way.*

2.3.5 Digital Signature Schemes

Digital signatures allow the holder of a secret key $\mathsf{sk}_{\mathsf{Sig}}$ to sign a message $m \in \mathcal{MS}$, while with knowledge of the corresponding public key $\mathsf{pk}_{\mathsf{Sig}}$ everyone can verify whether a given signature was actually endorsed by the signer, i.e., the holder of $\mathsf{pk}_{\mathsf{Sig}}$ [RSA78].

Definition 2.16 (Digital Signatures)**.** *A standard digital signature scheme* DSIG *consists of three algorithms* $\{\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Verify}_{\mathsf{Sig}}\}$ *such that:*

$\mathsf{KeyGen}_{\mathsf{Sig}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{Sig}}$ *outputs the public and private key of the signer, where* λ *is the security parameter:*

$$(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$$

$\mathsf{Sign}_{\mathsf{Sig}}$**.** *The algorithm* $\mathsf{Sign}_{\mathsf{Sig}}$ *gets as input the secret key* $\mathsf{sk}_{\mathsf{Sig}}$ *and the message* $m \in \mathcal{MS}$ *to sign. It outputs a signature:*

$$\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m)$$

$\mathsf{Verify}_{\mathsf{Sig}}$**.** *The deterministic algorithm* $\mathsf{Verify}_{\mathsf{Sig}}$ *outputs a decision bit* $d \in \{\mathsf{false}, \mathsf{true}\}$*, indicating if the signature* σ *is valid, w.r.t.* $\mathsf{pk}_{\mathsf{Sig}}$ *and* m*:*

$$d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m, \sigma)$$

Correctness. For each DSIG it is required that the correctness properties hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$, for all $m \in \mathcal{MS}$, $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m, \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m)) = \mathsf{true}$ is true. This definition captures perfect correctness.

Security. Two different security notions are used throughout this thesis. These are presented next.

Unforgeability against chosen-message attacks (UNF-CMA). Now, (weak) unforgeability of digital signature schemes [GMR88] is defined. In a nutshell, it is required that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with a signature σ^* for a *new* message m^*, i.e., for a message it has never seen a signature for. The adversary $\mathcal{A}$ can adaptively query for signatures on messages of its own choice.

Definition 2.17 (Unforgeability)**.** *A signature scheme* DSIG *is* ***unforgeable****, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{UNF\text{-}CMA}_{\mathcal{A}}^{\mathsf{DSIG}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.3.

Experiment $\mathsf{UNF\text{-}CMA}^{\mathsf{DSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}'_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \cdot)}}(\mathsf{pk}_{\mathsf{Sig}})$
where oracle $\mathsf{Sign}'_{\mathsf{Sig}}$ on input $\mathsf{sk}_{\mathsf{Sig}}$ and m:
set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
return $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m)$
return 1, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m^*, \sigma^*) = \mathsf{true} \ \wedge \ m^* \notin \mathcal{Q}$
return 0

Figure 2.3: DSIG Unforgeability

Experiment $\mathsf{sUNF\text{-}CMA}^{\mathsf{DSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}'_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, \cdot)}}(\mathsf{pk}_{\mathsf{Sig}})$
where oracle $\mathsf{Sign}'_{\mathsf{Sig}}$ on input $\mathsf{sk}_{\mathsf{Sig}}$ and m:
let $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m)$
set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma)\}$
return σ
return 1, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m^*, \sigma^*) = \mathsf{true} \ \wedge \ (m^*, \sigma^*) \notin \mathcal{Q}$
return 0

Figure 2.4: DSIG Strong Unforgeability

Strong Unforgeability against chosen-message attacks (sUNF-CMA). For some occasions, normal unforgeability of digital signature schemes is not enough, as the definition does not rule out that the adversary $\mathcal{A}$ can come up with *new* signatures for messages it knows signatures for [ADR02]. In a nutshell, strong unforgeability requires that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with any new message/signature pair (m^*, σ^*) which it has not seen before. As for (weak) unforgeability, the adversary $\mathcal{A}$ can adaptively query for signatures on messages of its own choice.

Definition 2.18 (Strong Unforgeability)**.** *A signature scheme* DSIG *is strongly unforgeable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\Pr[\mathsf{sUNF\text{-}CMA}^{\mathsf{DSIG}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.4.

Depending on the concrete use-case, different security properties are required. Thus, it is made explicit at the specific locations what security guarantees are needed from the used DSIG.

Examples. An example for a strongly unforgeable DSIG is RSA Full-Domain Hash (RSA-FDH) [BR96], introduced next, which is secure in the random-oracle model, if the RSA-Assumption holds [Cor00, KK12].

RSA-FDH Signatures. The RSA-FDH signature scheme $\mathsf{DSIG}_{\mathsf{RSA}} = (\mathsf{KeyGen}_{\mathsf{Sig}}, \mathsf{Sign}_{\mathsf{Sig}}, \mathsf{Verify}_{\mathsf{Sig}})$ associated to RSA-key generator RSAGen is defined as follows:

Construction 2.19 ($\mathsf{DSIG}_{\mathsf{RSA}}$). *It requires a hash function $\mathcal{H}_{\mathsf{RSA}} : \{0,1\}^* \to \mathbb{Z}_N^*$, modeled as a random oracle.*

$\mathsf{KeyGen}_{\mathsf{Sig}}$. *The algorithm $\mathsf{KeyGen}_{\mathsf{Sig}}$ generates the key pair in the following way:*

1. *Run $(N, e, d, p, q) \xleftarrow{\$} \mathsf{RSAGen}(1^\lambda)$.*
2. *Output $((d, p, q), (N, e))$, where $N = pq$.*

$\mathsf{Sign}_{\mathsf{Sig}}$. *The algorithm $\mathsf{Sign}_{\mathsf{Sig}}$ generates a signature on $m \in \{0,1\}^*$ in the following way:*

1. *Return σ, where $\sigma \leftarrow (\mathcal{H}_{\mathsf{RSA}}(m))^d \mod N$.*

$\mathsf{Verify}_{\mathsf{Sig}}$. *The algorithm $\mathsf{Verify}_{\mathsf{Sig}}$ verifies a signature in the following way:*

1. *Return true, if $\mathcal{H}_{\mathsf{RSA}}(m) = \sigma^e \mod N$.*
2. *Return false.*

Sometimes, a signature scheme also requires an additional parameter generation algorithm. If this is required, it is made explicit.

2.3.6 Public-Key Encryption Schemes

Public-key encryption allows to encrypt a message m using a given public key $\mathsf{pk}_{\mathsf{ENC}}$. In a nutshell, the given ciphertext leaks no information of the contained message, except its length, if the corresponding secret key $\mathsf{sk}_{\mathsf{ENC}}$ is not known.

Definition 2.20 (Labeled Public-Key Encryption Schemes). *A labeled public-key encryption scheme ENC consists of four algorithms $\{\mathsf{PPGen}_{\mathsf{ENC}}, \mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Dec}_{\mathsf{ENC}}\}$, such that:*

$\mathsf{PPGen}_{\mathsf{ENC}}$. *The algorithm $\mathsf{PPGen}_{\mathsf{ENC}}$ outputs the public parameters of the scheme:*

$$\mathsf{pp}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{ENC}}(1^\lambda)$$

It is assumed that $\mathsf{pp}_{\mathsf{ENC}}$ is implicit input to all other algorithms. Also, this algorithm may be omitted, if it is clear from the context.

$\mathsf{KeyGen}_{\mathsf{ENC}}$. *The algorithm $\mathsf{KeyGen}_{\mathsf{ENC}}$ outputs the public and private key, on input $\mathsf{pp}_{\mathsf{ENC}}$:*

$$(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(\mathsf{pp}_{\mathsf{ENC}})$$

Experiment $\mathsf{IND\text{-}CPA}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda)$:
$\mathsf{pp}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{ENC}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(\mathsf{pp}_{\mathsf{ENC}})$
$b \xleftarrow{\$} \{0,1\}$
$((m_0^*, m_1^*), state_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}(\cdot)}}(\mathsf{pk}_{\mathsf{ENC}})$
If $|m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{MS} \vee m_1^* \notin \mathcal{MS}$:
  $c^* \leftarrow \perp$
Else:
  $c^* \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m_b^*)$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}(\cdot)}}(state_{\mathcal{A}}, c^*)$
return 1, if $a = b$
return 0

Figure 2.5: ENC IND-CPA Security

$\mathsf{Enc}_{\mathsf{ENC}}$. *The algorithm* $\mathsf{Enc}_{\mathsf{ENC}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{ENC}}$*, the message* $m \in \mathcal{MS}$ *to encrypt, and also some label* $\ell \in \{0,1\}^*$*. It outputs a ciphertext:*

$$c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell)$$

$\mathsf{Dec}_{\mathsf{ENC}}$. *The deterministic algorithm* $\mathsf{Dec}_{\mathsf{ENC}}$ *outputs a message* m *(or* $\perp$*, if the ciphertext is invalid) on input* $\mathsf{sk}_{\mathsf{ENC}}$*, label* ℓ *and a ciphertext* c*:*

$$m \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, \ell)$$

This defines labeled public-key encryption schemes. For a definition without labels, one can simply fix ℓ to the empty string ϵ, also for the security definitions. Sometimes, $\mathsf{PPGen}_{\mathsf{ENC}}$ is omitted, if clear from the context.

Correctness. For each ENC, the usual correctness properties must hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{ENC}}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(\mathsf{pp}_{\mathsf{ENC}})$, for all $m \in \mathcal{MS}$, and for all $\ell \in \{0,1\}^*$, $\mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell), \ell) = m$ is true. Again, this definition captures perfect correctness.

Security. For encryption schemes, a plethora of different notions appeared in the literature. For now, only two definitions are required. Additional notions are introduced in Chapter 4 and Chapter 5.

Chosen plaintext attacks (IND-CPA). The following definition is derived from the ideas given by Goldwasser and Micali [GM84]. Note, this definition has no label ℓ.

Experiment $\mathsf{IND\text{-}CCA2}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda)$:
$\mathsf{pp}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{ENC}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(\mathsf{pp}_{\mathsf{ENC}})$
$b \xleftarrow{\$} \{0,1\}$
$((m_0^*, m_1^*), \ell^*, state_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}(\cdot)}, \mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, \cdot, \cdot)}}(\mathsf{pk}_{\mathsf{ENC}})$
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input $\mathsf{sk}_{\mathsf{ENC}}$, c and ℓ:
return $m' \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, \ell)$
where $\mathcal{O}^{\mathcal{H}(\cdot)}$ on input s:
return $\mathcal{H}(s)$
If $|m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{MS} \vee m_1^* \notin \mathcal{MS}$:
$c^* \leftarrow \perp$
Else:
$c^* \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m_b^*, \ell^*)$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}(\cdot)}, \mathcal{O}^{\mathsf{Dec}'_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, \cdot, \cdot)}}(state_{\mathcal{A}}, c^*)$
where oracle $\mathsf{Dec}'_{\mathsf{ENC}}$ behaves as $\mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, \cdot, \cdot)}$,
but returns $\perp$ if (c^*, ℓ^*) is queried.
return 1, if $a = b$
return 0

Figure 2.6: ENC Labeled IND-CCA2 Security

Definition 2.21 (IND-CPA-Security)**.** *An encryption scheme* ENC *is IND-CPA-secure, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{IND\text{-}CPA}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.5.

Adaptive chosen-ciphertext attacks (IND-CCA2). The following definition is derived from Camenisch and Shoup, as well as from Naor and Yung [CS03, NY90].

Definition 2.22 (Labeled IND-CCA2-Security)**.** *A labeled encryption scheme* ENC *is IND-CCA2-secure, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{IND\text{-}CCA2}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 2.6.

Note, both security definitions already have explicit access to a random oracle. If this is not wanted, one can set $\mathcal{H} = \perp$. Why this is made explicit in this definition is discussed in Chapter 4.

2.3.7 Non-Interactive Commitment Schemes

Non-interactive commitment schemes allow one party to commit itself to a value without revealing it [Blu81]. Later, the committing party can give some opening information to the receiver, which can then "open" the commitment. This corresponds to a safe containing a letter, where the key can be given to the receiver of the safe at some later point.

Definition 2.23 (Non-Interactive Commitments)**.** *A non-interactive commitment scheme* CC *consists of three PPT algorithms* $\{\mathsf{ParGen}_{\mathsf{CC}}, \mathsf{Commit}_{\mathsf{CC}}, \mathsf{Open}_{\mathsf{CC}}\}$*, such that:*

$\mathsf{ParGen}_{\mathsf{CC}}$**.** *This algorithm takes as input a security parameter* λ *and outputs the public parameters* $\mathsf{pp}_{\mathsf{CC}}$*:*

$$\mathsf{pp}_{\mathsf{CC}} \xleftarrow{\$} \mathsf{ParGen}_{\mathsf{CC}}(1^\lambda)$$

$\mathsf{Commit}_{\mathsf{CC}}$**.** *This algorithm takes as input a message* m *and outputs a commitment* C *together with corresponding opening information* O*:*

$$(C, O) \xleftarrow{\$} \mathsf{Commit}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, m)$$

$\mathsf{Open}_{\mathsf{CC}}$**.** *This deterministic algorithm takes as input a commitment* C *with corresponding opening information* O *and outputs message* $m \in \mathcal{MS}$*:*

$$m \leftarrow \mathsf{Open}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, C, O)$$

Correctness. A commitment scheme CC is said to be correct, if for all $\lambda \in \mathbb{N}$, all $\mathsf{pp}_{\mathsf{CC}} \xleftarrow{\$} \mathsf{ParGen}_{\mathsf{CC}}(1^\lambda)$, for all messages $m \in \mathcal{MS}$, for all $(C, O) \xleftarrow{\$} \mathsf{Commit}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, m)$, it holds that $\mathsf{Open}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, C, O) = m$. This captures perfect correctness.

Security. Two different security notions are required. These are presented next.

Binding. Binding means that a commitment cannot be opened to two different messages.

Definition 2.24 (Binding)**.** *A non-interactive commitment scheme is binding, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr\left[\begin{array}{c} \mathsf{pp}_{\mathsf{CC}} \xleftarrow{\$} \mathsf{ParGen}_{\mathsf{CC}}(1^\lambda), (C^*, O^*, O'^*) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\mathsf{CC}}), \\ m \leftarrow \mathsf{Open}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, C^*, O^*), m' \leftarrow \mathsf{Open}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, C^*, O'^*) : \\ m \neq m' \ \wedge \ m \neq \bot \ \wedge \ m' \neq \bot \end{array}\right] \leq \nu(\lambda)$$

If $\nu(\lambda) = 0$*, even for unbounded adversaries, a commitment scheme is said to be perfectly binding.*

Hiding. Hiding means that a commitment does not leak which message it contains, as long as the corresponding opening information is not known.

Definition 2.25 (Hiding)**.** *A non-interactive commitment scheme is hiding, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr\left[\begin{array}{c}(\mathsf{pp}_{\mathsf{CC}}, m_0, m_1, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(1^\lambda), b \xleftarrow{\$} \{0,1\},\\ (C, O) \xleftarrow{\$} \mathsf{Commit}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, m_b), b^* \xleftarrow{\$} \mathcal{A}(C, \mathsf{state}) :\\ b = b^*\end{array}\right] - \frac{1}{2}\right| \leq \nu(\lambda)$$

If $\nu(\lambda) = 0$, even for unbounded adversaries, the commitment scheme is said to be perfectly hiding. Note, the adversary can even generate the parameters for the CC*.*

Definition 2.26 (Secure CCs)**.** *A commitment scheme* CC *is secure, if it is correct, binding and hiding.*

Note, a commitment scheme CC cannot be perfectly hiding and perfectly binding at the same time [FF11]. An example for perfectly-hiding commitment-schemes are Pedersen commitments [Ped91], which are binding under the DL-Assumption. An example for perfectly-binding commitment-schemes are perfectly correct CPA-secure encryption schemes, e.g., El Gamal [Gam84], which is hiding under DDH.

2.4 Universal Composition – In a Nutshell

As already discussed in Chapter 1, game-based security definitions are easier to understand for simple primitives. However, they have also some serious drawbacks. Namely, they become pretty hard to understand for more complex systems far beyond digital signatures. Moreover, they do not guarantee that they interact well with other primitives in a composed system, or in some other higher-level protocol, if not defined very carefully. As a prominent example, one can consider interactive zero-knowledge proofs [GMR85], which may be trivially insecure, if run in parallel [GK96].

It is therefore no surprise that one wants to have a framework which guarantees that once security of a certain primitive is proven, it remains secure in arbitrary contexts and can then be used as a given building block in other protocols, without the need to re-prove the whole system. This is known as *universal composition* (UC). The best-known framework is Canetti's UC-framework [Can01], even though related frameworks with similar goals have been published [HS15, Küs06, PW01, Wik16]. In this thesis, the UC framework by Canetti is used, as it seems to be the current standard, even though it has some serious flaws, e.g., the runtime definition [HUM13]. However, from a purely practical perspective, they seem to be of almost no real-life relevance [End16].

In more detail, the UC-framework [Can01] enables the modular design of cryptographic protocols by analyzing the security of composed protocols in a so-called "hybrid" model, where subprotocols are replaced by their "ideal functionalities", thereby eliminating the need for explicit reductions from the security of the building blocks in the overall security proof. In a nutshell, an ideal functionality $\mathcal{F}$ describes what input and output interfaces a protocol offers to the parties, but also which power the adversary has and which information the adversary

receives. Thus, an ideal functionality $\mathcal{F}$ can be seen as a trusted third party which receives input from all parties and also calculates and hands back the results in a perfectly secure manner. Therefore, an ideal functionality $\mathcal{F}$ specifies the security and correctness of a protocol at the same time. This idea dates back to Goldwasser et al. [GMW87]. The input for the participating parties are provided by an abstract entity called the environment $\mathcal{Z}$, which is modeled as an interactive Turing machine. The environment $\mathcal{Z}$ can also freely communicate with the real-world adversary $\mathcal{A}$, which is also modeled as an interactive Turing machine. The adversary $\mathcal{A}$ interacts with the protocol in question directly, incorporating the network, but also any corrupted parties. To distinguish between different instances of the protocol, UC requires that a globally unique session identifier *sid* is assigned. All honest parties participating in an instance with a given *sid* execute the same protocol, while neither the adversary's nor the environment's code are specified. Each party also receives a unique party identifier *pid* to distinguish between the different entities participating in the protocol. It is also important to note that in UC at most one party has control, i.e., neither computation nor network traffic is concurrent.

Faithful to its name, the UC framework guarantees that any secure instantiation of the (sub)protocols yields a secure instantiation in the composed protocol, even though the composed protocol may be insecure. To prove that a given protocol realizes an ideal functionality, one constructs an ideal world adversary SIM (the "simulator") which interacts with the ideal functionality and the real adversary such that, informally speaking, for *any* environment $\mathcal{Z}$ and any (arbitrary but fixed) adversary $\mathcal{A}$ the view of real execution is indistinguishable from the view the same environment receives in the ideal world run with the simulator SIM. In other words, UC requires that for every attack by the adversary $\mathcal{A}$ in the real world, SIM can translate the attack into the ideal world. Hence, a protocol is said to *emulate* a given functionality $\mathcal{F}$, as the real adversary cannot do any more harm as defined and is thus already incorporated into the definition. More formally, there is the following definition.

Definition 2.27. *A protocol π UC-emulates a protocol ϕ, if for all (non-uniform) PPT adversaries $\mathcal{A}$ there exists a (non-uniform) PPT simulator SIM such that for all (non-uniform) environments $\mathcal{Z}$ the following holds:*

$$\mathsf{Exec}(\pi, \mathcal{A}, \mathcal{Z})(1^\lambda, z) \approx_c \mathsf{Exec}(\phi, \mathsf{SIM}, \mathcal{Z})(1^\lambda, z)$$

$\mathsf{Exec}(\pi, \mathcal{A}, \mathcal{Z})(1^\lambda, z)$ is defined as the distribution ensemble $\{\mathsf{Exec}(\pi, \mathcal{A}, \mathcal{Z})(1^\lambda, z)\}_{\lambda\in\mathbb{N}}$, while $\mathsf{Exec}(\phi, \mathsf{SIM}, \mathcal{Z})(z)$ is defined as the distribution ensemble $\{\mathsf{Exec}(\phi, \mathsf{SIM}, \mathcal{Z})(1^\lambda, z)\}_{\lambda\in\mathbb{N}}$. The variable $z \in \{0,1\}^*$ is a polynomial-size advice for the non-uniform model of computation.

Note, if a proof is given in the uniform model of computation, it is also valid in the non-uniform setting, i.e., if the advice (here z) is ignored, the proofs are still correct. The converse is not true in general [CLMP13, KM13].

Universal composition now allows to derive the following theorem, which is the whole point of defining and using UC-like models:

Theorem 2.28. *Let π, ϕ, ρ be some protocols such that π UC-emulates ϕ, while ρ uses π as a subprotocol. Then $\rho^{\phi\to\pi}$ emulates ρ.*

Here, $\rho^{\phi\to\pi}$ means that the calls by ρ to the subprotocol π are replaced with calls to the protocol ϕ. In other words, if π UC-realizes an ideal functionality $\mathcal{F}$, while a protocol makes use of $\mathcal{F}$, it is save to replace $\mathcal{F}$ with π, which is exactly the hybrid-model of computation.

2.4.1 Corruption Models

Canetti defined three corruption models [Can01]. Namely, these are static corruptions, dynamic corruptions and transient corruptions (which is a subset of dynamic corruptions). These have later been refined to account for a more fine-grained model. Next, a quick overview over the different corruption models is given, as this thesis deals with different ones.

2.4.1.1 Static Corruptions

If a protocol is proven secure assuming static corruptions, the adversary has to corrupt the parties it wants to control *before* the protocol starts. Clearly, this corruption model only gives feeble security guarantees. However, depending on the use-case, it may still be strong enough and sometimes statically secure protocols can be lifted to support adaptive corruptions [BFSK11].

2.4.1.2 Dynamic Corruptions

Dynamic corruptions allow the adversary to corrupt a party after the protocol has started and has already run for an adversarially chosen amount of time. Upon corruption, the adversary wants to see a convincing state of the corrupted party, e.g., randomness, an execution history, secret keys and the like. Note, the environment knows which parties are corrupted, thus prohibiting the simulator corrupting all parties and to achieve its task in a trivial way [Can01]. Depending what assumptions are reasonable, this corruption model must further be divided.

With Erasures. If one assumes secure erasures, parties can erase parts of the state. For example, if a party deletes the randomness used to generate a key pair, the adversary is "satisfied" if it only sees the secret key and not the randomness used to create it. Refer to Chapter 4 for additional examples. Similar arguments apply, e.g., for zero-knowledge proofs [CKS11]. This may allow to prove certain instantiations secure, which may not provably secure otherwise. However, assuming secure erasures may not always be acceptable, as it may be very hard to achieve [CCGS10]. Moreover, depending on what security guarantees one requires, this corruption model has two subtypes.

Receiving Prior Input/Output. This is the normal definition. Here, the simulator SIM receives the prior input/output of the corrupted party and can construct the state of the corrupted party based on the now learned information.

Not Receiving Prior Input/Output. This is the opposite. Here, the simulator does not receive the prior input/output of that party. Clearly, this corruption model implies secure erasures, as otherwise the adversary expects all input and output somewhere in the execution history. This corruption model is introduced in Chapter 4.

Clearly, one could also define "intermediate" stages, where the simulator only needs to simulate fractions (say, the last five algorithms run) of the execution history, or only 50% of the execution history. These intermediate definitions are not used in this thesis, but may lead to interesting new insights, much like non-perfectly erasable memory [CEM16].

Without Erasures. Security without secure erasures is the strongest corruption model considered in this thesis. Here, the adversary expects, upon corruption, the complete state and a convincing execution history. Thus, it is obvious the simulator SIM receives all prior input/output of the now corrupted party.

Each of the four aforementioned corruption models can also support transient corruptions. Here, corrupt entities can recover from corruption after a special input from the environment. However, only a handful of protocols actually achieving this notion have been presented so far [CEN15, CLN15]. This thesis does not deal with this kind of corruption.

2.4.2 General Conventions

When writing ideal functionalities and protocols, the following conventions are used. If additional conventions are necessary, this is made explicit at the specific locations.

- If a party receives a network message which does correspond to a message format expected, it is ignored, while the environment receives an implicit error message.
- If a functionality receives input for which it does not have a corresponding interface, it outputs an implicit error message directly to the environment, i.e., it also gives up control. The same is true for all protocol machines.
- For some protocols, responsive environments and adversaries are required [CEK+16]. In a nutshell, this means that an adversary or environment needs to directly answer a request for a given interface. This is necessary, if certain "meta-information" is required. In more detail, the notation "send x to $\mathcal{A}$ and wait for y from $\mathcal{A}$" is used as a shorthand notation for requests to responsive environments/adversaries as defined by Camenisch et al. [CEK+16], such that the functionality "stalls" until $\mathcal{A}$ provides a response y through a dedicated interface. While the functionality waits for a message from $\mathcal{A}$, the adversary/environment cannot invoke any other interfaces of the ideal functionality, generate any network traffic or activate/corrupt any other parties.
- An output is "delayed", if the simulator can decide when the party actually outputs a message. The output can either be private, where the adversary does not learn its content, or public, where the adversary leans the message. "Hybrid" definitions, i.e., where some parts are public and some are not, are not used in this thesis.

2.5 The Random-Oracle Methodology

In a nutshell, a random oracle (RO) is a purely random function which can be called by all participants, including the adversary [BR93]. Random oracles are normally used to abstract from, and idealize, real hash-functions (which are not random) such as SHA-3 [BDPA14]. Thus, a random oracle can be seen as a black-box where no participant can look "into". This is necessary, as a purely random function requires an exponential amount of space to be stored, which is clearly impossible in a PPT setting [Mit16]. These random oracles help, e.g., in security proofs, when the challenger takes over control of the random oracle, simulating it for the adversary. This possibility can be used to embed certain challenges into the random-oracle responses, and to "see" the queries made by the adversary, i.e., to extract pre-images, which thus is a very strong model [AB13, BF11a, FLR+10]. In particular, there are impossibility

1. `Query.` Upon input (QUERY, sid, m), $m \in \mathcal{D}$, from a party $\mathcal{P}$ do:
   - If there is a tuple $(m, \hat{h}) \in \mathcal{L}$ for some $\hat{h}$, let $h \leftarrow \hat{h}$.
   - Else, draw $\hat{h} \xleftarrow{\$} \mathcal{R}$, add $(m, \hat{h})$ to $\mathcal{L}$, and set $h \leftarrow \hat{h}$.
   - Output $(\mathsf{RESPONSE}, sid, m, h)$ to $\mathcal{P}$.

Figure 2.7: Ideal random-oracle functionality $\mathcal{F}_{\mathsf{RO}}^{\mathcal{D}\to\mathcal{R}}$, where $\mathcal{L}$ is an initially empty list

results that state that random oracles cannot be instantiated [CGH04]. Namely, there are constructions which are secure in the random-oracle model, while *any* instantiation without random oracles make the construction trivially insecure. On the upside, however, there is not a single, not on purpose broken, cryptographic scheme which has been succesfully attacked in the real world [KM15]. Moreover, the random-oracle methodology [BR93] has proven useful in many scenarios, including, but not limited to, NIZKs [FS86], digital signatures [Cor00] and identity-based encryption [BF01]. Thus, the random-oracle methodology needs to be treated as a heuristic, but has its merits.

Unfortunately, there are results that prove that certain primitives cannot be instantiated in the computational model, e.g., non-interactive secure message transfer [Nie02]. Thus, recently published results, including UCEs and indistinguishability obfuscation, cannot be used in all constructions [BHK13, BFM14, HSW14]. This is also true for some of the constructions given in this thesis.

However, there is the additional benefit that primitives which make use of the random-oracle methodology tend to be more efficient than schemes in the standard, i.e., computational model of computation. The random-oracle methodology is used in several occasions presented in this thesis to side-step certain impossibility results, but also to realize efficient and practical schemes beyond the PPT-barrier.

Mittelbach's PhD thesis provides a deeper and more formal insight into the random oracle methodology, its possibilities, its shortcomings, but also how to circumvent certain restrictions [Mit16].

2.6 Some Standard Ideal Functionalities

Some of the protocols introduced in this thesis make use of some already defined functionalities. To avoid duplicate work, but to be complete, these functionalities are given here. These include the random-oracle functionality $\mathcal{F}_{\mathsf{RO}}^{\mathcal{D}\to\mathcal{R}}$, the ideal certificate authority $\mathcal{F}_{\mathsf{CA}}$, and the authenticated channel functionality $\mathcal{F}_{\mathsf{Auth}}$.

2.6.1 Ideal Random-Oracle Functionality $\mathcal{F}_{\mathsf{RO}}^{\mathcal{D}\to\mathcal{R}}$

Figure 2.7 shows the random-oracle functionality $\mathcal{F}_{\mathsf{RO}}^{\mathcal{D}\to\mathcal{R}}$, derived from Hofheinz and Müller-Quade [HM04]. In a nutshell, every party queries the functionality, which then returns a completely random value, if not already assigned. The functionality is parametrized with a domain $\mathcal{D}$ and a range $\mathcal{R}$. Note, this is a single instance functionality. If multiple instances are required, different session identifiers can be used.

1. `Registration.` Upon input $(\mathsf{REGISTER}, sid, v)$ from a party $\mathcal{P}$:
   - Send $(\mathsf{REGISTER}, sid, v)$ to $\mathcal{A}$. Upon receiving ok from $\mathcal{A}$, and if $sid = (\mathcal{P}, sid')$, and this is the first request from $\mathcal{P}$, record $(\mathsf{pki\text{-}rec}, sid, v)$.
2. `Retrieve.` Upon receiving a message $(\mathsf{RETRIEVE}, sid)$ from a party $\mathcal{P}'$:
   - Send $(\mathsf{RETRIEVE}, sid, \mathcal{P}')$ to $\mathcal{A}$. Upon receiving ok from $\mathcal{A}$:
     - If a record $(\mathsf{pki\text{-}rec}, sid, v)$ exists, output $(\mathsf{pki\text{-}rec}, sid, v)$ to $\mathcal{P}'$.
     - Else, output $(\mathsf{pki\text{-}rec}, sid, \perp)$ to $\mathcal{P}'$.

Figure 2.8: Ideal certificate authority functionality $\mathcal{F}_{\mathsf{CA}}$

1. `Send.` Upon input (SEND, sid, m) from party $\mathcal{S}$:
   - Check that $sid = (\mathcal{S}, \mathcal{R}, sid')$.
   - Send public delayed output $(\mathsf{SEND}, sid, \mathcal{S}, m)$ to $\mathcal{R}$, if $\mathcal{R}$ has not received any output yet.
2. `Corrupt.` Upon receiving a message $(\mathsf{CORRUPT}, sid, m')$ from $\mathcal{A}$:
   - Check that $sid = (\mathcal{S}, \mathcal{R}, sid')$.
   - If $\mathcal{S}$ is not corrupt, ignore.
   - Output $(\mathsf{SEND}, sid, \mathcal{S}, m')$ to $\mathcal{R}$, if $\mathcal{R}$ has not received any output yet.

Figure 2.9: Ideal authenticated-channel functionality $\mathcal{F}_{\mathsf{Auth}}$

2.6.2 Ideal Certificate Authority $\mathcal{F}_{\mathsf{CA}}$

The ideal certificate authority functionality $\mathcal{F}_{\mathsf{CA}}$, depicted in Figure 2.8, is the UC-pedant of a PKI, i.e., as some kind of "authenticated bulletin-board" [Can04]. This functionality is used to register public keys in an authenticated fashion.

2.6.3 Ideal Authenticated Channel $\mathcal{F}_{\mathsf{Auth}}$

Next, the authenticated channel functionality $\mathcal{F}_{\mathsf{Auth}}$ is presented, derived from Canetti [Can04]. In a nutshell, this functionality allows to send a *single public* message m from the sender $\mathcal{S}$ to the receiver $\mathcal{R}$. Multiple messages can be sent using multiple instances of this functionality. The same is true for bi-directional transfers, i.e., $\mathcal{R}$ and $\mathcal{S}$ change roles. Also note, that the receiver and the sender are encoded into the *sid*, while the adversary can only overwrite messages which have not yet been delivered.

2.6.4 Ideal Signatures $\mathcal{F}_{\mathsf{Sig}}$

Depicted in Figure 2.10 is the definition given by Canetti [Can04], but adjusted for the used notation, already incorporating the observations by Backes and Hofheinz [BH04]. Namely, the key generation is modeled more explicitly, while the functionality also imposes strong unforgeability. Also, the adversary does not learn which party verifies a signature, which is not the case in already proposed functionalities.

1. `Key Generation.` Upon input (KEYGEN, sid) of party $\mathcal{P}$:
   - Check that $sid = (\mathcal{P}, sid')$. If not, ignore.
   - If there is a record $(\mathsf{key\text{-}rec}, sid, \cdot)$, ignore.
   - Send (KEYGEN, sid) to $\mathcal{A}$, and wait for input $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$ from $\mathcal{A}$.
   - If $\mathsf{pk}_{\mathsf{Sig}} = \perp$, ignore.
   - Create record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}})$.
   - Output $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$ to $\mathcal{P}$.
2. `Signature Generation Request.` Upon input (SIGN, sid, m) from $\mathcal{P}$:
   - Check that $sid = (\mathcal{P}, sid')$. If not, ignore.
   - If there is no record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}})$, ignore.
   - Send (SIGN, sid, m) to $\mathcal{A}$, and wait for input $(\mathsf{SIGNATURE}, sid, m, \sigma)$ from $\mathcal{A}$.
   - If there is a record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{false})$, ignore.
   - Create record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{true})$.
   - Output $(\mathsf{SIGNATURE}, sid, m, \sigma)$ to $\mathcal{P}$.
3. `Verification.` Upon input $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}})$ from party $\mathcal{P}'$:
   - If $\sigma = \perp$, output $(\mathsf{VERIFY}, sid, m, \perp, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{false})$ to $\mathcal{P}'$.
   - If there is a record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, v)$, output $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, v)$ to $\mathcal{P}'$.
   - If $\mathcal{P}$ is honest, and there is a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}'_{\mathsf{Sig}})$, with $sid = (\mathcal{P}, sid')$, create record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{false})$, and output $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{false})$ to $\mathcal{P}'$.
   - Send $(\mathsf{SIGVERF}, sid, m, \mathsf{pk}'_{\mathsf{Sig}}, \sigma)$ to $\mathcal{A}$, and wait for input $(\mathsf{SIGVERF}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \phi)$ from $\mathcal{A}$.
   - If $\phi \notin \{\mathsf{true}, \mathsf{false}\}$, ignore.
   - If $\mathsf{pk}'_{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{Sig}}$ taken from $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}})$, and $\mathcal{P}$ is honest, where $sid = (\mathcal{P}, sid')$, and there is no record $(\mathsf{sig\text{-}rec}, sid, m, \cdot, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{true})$, set $\phi \leftarrow \mathsf{false}$.
   - Create record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \phi)$, and output $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \phi)$ to $\mathcal{P}'$.

Figure 2.10: Ideal signature functionality $\mathcal{F}_{\mathsf{Sig}}$

2.6.4.1 Explanation

Now, a description what each interface actually does is given, as this functionality has been altered.

1. The KEYGEN interface allows $\mathcal{P}$ to generate a key pair. This duty is left to the adversary which needs to provide a public key. This interface only passes the key generation request on the first call.
2. The SIGN interface allows a party to request a signature on a message m. This duty is given to the adversary.
3. The VERIFY interface allows a party to verify the validity of a given signature. It is required that the party also specifies the public key. As long as the generating party is honest, the functionality can directly determine the validity of the signature under the stored $\mathsf{pk}_{\mathsf{Sig}}$, as it imposes strong unforgeability. The functionality also imposes consistency, i.e., a valid signature remains valid, while a signature once marked as non-valid will never become valid. If the functionality cannot determine the validity of the signature itself, it asks the adversary to do so.

2.6.4.2 Construction

The construction is a wrapper around any strongly unforgeable signature scheme to match the input and output behavior of the ideal functionality $\mathcal{F}_{\mathsf{Sig}}$. Note, the *sid* is added to the message to be signed to bind a signature to a session *sid*.

Construction 2.29 (UC-Secure Digital Signatures)**.** *The participants react on certain inputs received from the environment. In particular, a signature scheme which is UC-secure can be constructed as follows, assuming* DSIG *is a strongly unforgeable signature scheme.*

Key Generation: *On input of* (KEYGEN, sid)*, check that* $sid = (\mathcal{P}, sid')$*. If this is not the case, ignore. If a record* $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}})$ *exists, ignore. Generate* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*. Store* $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}})$*. Output* $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$*.*

Signature Generation: *On input of* (SIGN, sid, m)*, ignore, if no record* $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}})$ *exists. Generate* $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, m))$*. Output* $(\mathsf{SIGNATURE}, sid, m, \sigma)$*.*

Verification: *On input of* $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}})$*, let* $d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}'_{\mathsf{Sig}}, (sid, m), \sigma)$*. Output* $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, d)$*.*

Theorem 2.30. *The protocol given above is a secure realization of the ideal functionality* $\mathcal{F}_{\mathsf{Sig}}$ *given in Figure 2.10 without secure erasures and adaptive corruptions.*

Proof. To prove security, a simulator SIM which acts in the ideal world and translates between the real adversary and $\mathcal{F}_{\mathsf{Sig}}$ must be provided. Clearly, an environment only sees a difference between the real world and the ideal world, if the adversary was able to generate a signature σ^* which verifies under $\mathsf{pk}_{\mathsf{Sig}}$ which was never created by the honest party, ignoring adaptive corruptions. A standard reduction yields an adversary which breaks the strong unforgeability of the used signature scheme.

2.6.4.3 Simulator

Subsequently the simulator SIM is described. Note, SIM is never "exposed" to $\mathcal{A}$ directly, as the "network traffic" is part of the environment, while corrupted parties are incorporated into $\mathcal{A}$. Of course, $\mathcal{A}$ may still corrupt parties. SIM also does not know the party identities requesting verifications and thus also does not know what party to simulate. Thus, each request is treated independently, while the connection to the corresponding party is only made at corruption.

Key Generation. The simulator SIM gets activated on input (KEYGEN, sid). It simulates "$\mathcal{P}$"completely honestly. If "$\mathcal{P}$"outputs $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$, SIM also sends $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$ to $\mathcal{F}$.

Signature Generation. Here, the simulator SIM receives input (SIGN, sid, m). It simulates "$\mathcal{P}$"completely honestly. If "$\mathcal{P}$"outputs $(\mathsf{SIGNATURE}, sid, m, \sigma)$, SIM sends $(\mathsf{SIGNATURE}, sid, m, \sigma)$ to $\mathcal{F}_{\mathsf{Sig}}$.

Signature Verification. Here, the simulator SIM receives input $(\mathsf{SIGVERF}, sid, m, \mathsf{pk}'_{\mathsf{Sig}}, \sigma)$. The simulator simply calculates $d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}'_{\mathsf{Sig}}, (sid, m), \sigma)$. It then sends $(\mathsf{SIGVERF}, sid, m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, d)$ to $\mathcal{F}_{\mathsf{Sig}}$. Note, SIM does not learn which party wants to verify a signature, and can thus not assign this execution to any party yet.

Corruption. Now is described how corruptions are handled. Once more, the standard corruption definition given by Canetti [Can01] is used. In particular, upon corruption of a party, the simulator receives all inputs and outputs of that party and then needs to provide a consistent view. Two different cases need to be considered. Note, the indices are omitted to have an easier to read representation.

Corruption of the Holder of the Secret Key ($\mathcal{P}$). If the holder of the secret key becomes corrupted, SIM receives six lists corresponding to the inputs and outputs of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. SIM ignores all of them but the ones of the form $(\mathsf{VERIFY}, sid, \sigma, m, \mathsf{pk}'_{\mathsf{Sig}})$. As signature generation was done honestly, "$\mathcal{P}$"is already in the correct state, including randomness. For signature verification, however, SIM only learns now which signatures "$\mathcal{P}$" was verifying. As verification is deterministic, SIM can simply run $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}'_{\mathsf{Sig}}, (sid, m), \sigma)$ for each signature and make that execution part of "$\mathcal{P}$".

Corruption of any Other Party $\mathcal{Q}$. Again, SIM receives six lists corresponding to the inputs and outputs of $\mathcal{F}_{\mathsf{Sig}}$. SIM ignores all of them but the ones of the form $(\mathsf{VERIFY}, sid, \sigma, m, \mathsf{pk}'_{\mathsf{Sig}})$ (Bogus key and signature generation requests do not matter, as they are simply ignored. Thus, the execution history is implicit). For signature verification SIM only learns now which signature "$\mathcal{Q}$" was verifying. As verification is deterministic, SIM can now simply run $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}'_{\mathsf{Sig}}, (sid, m), \sigma)$ for each signature and make that execution part of "$\mathcal{Q}$". □

This is a new proof, as no secure erasures are assumed, and thus also the execution history needs to be simulated, while the simulator does not learn which party verifies a signature.

2.6.5 Common Reference Functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{CRS}}$

Here, the common reference functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{CRS}}$, depicted in Figure 2.11, derived from Canetti and Fischlin [CF01], is introduced. Here, $\mathcal{D}$ is some distribution used to parametrize the functionality. In a nutshell, this functionality draws a common reference string CRS according to the distribution $\mathcal{D}$ on the first call. All other invocations then return the drawn element.

1. `Query`. Upon input $(\mathsf{RETRIEVE}, sid)$, from a party $\mathcal{P}$:
   - If there is no record $(sid, \mathsf{crs\text{-}rec}, \mathsf{CRS})$, draw $\mathsf{crs\text{-}rec} \xleftarrow{\$} \mathcal{D}$, and create record $(sid, \mathsf{crs\text{-}rec}, \mathsf{CRS})$.
   - Output $(\mathsf{GETCRSR}, sid, \mathsf{CRS})$, where CRS is taken from record $(sid, \mathsf{crs\text{-}rec}, \mathsf{CRS})$.

Figure 2.11: Common Reference String functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{CRS}}$

Chapter 3

UC-Secure Distributed Password-Based Single Sign-On

The results of this chapter are currently under submission.

Abstract. Single Sign-On (SSO) allows a user to conveniently authenticate to multiple service providers with only a single login credential, typically being a username and a password. Current solutions rely on a single ticket-granting server to verify the users' password and issue authentication tickets to the individual service provider. This approach crucially relies on the ticket-granting server being fully trusted: if corrupt, it can impersonate the users towards all service providers and offline attack the users' passwords. In fact, even if the ticket-granting server is honest but suffers from a data breach that exposes the password verification information, the attacker can brute-force the passwords and gain access to the users' accounts on all service providers. To avoid such a single point of failure while preserving the convenience of SSO, this chapter proposes the concept of password-based distributed single sign-on. Therein, still only a single user login is needed, but password verification and ticket-issuance are distributed over multiple ticket-granting servers. Distributed SSO prevents impersonation and offline attack against users' passwords as long as not all ticket-granting servers are corrupt. This is modeled and proven secure in the Universal Composability (UC) framework, which naturally captures password-specific issues such as password-typos and guessing passwords by an adversary. Two efficient and provably secure protocols are presented: the first follows the standard SSO concept where users are known under a single identity towards all service providers and all their actions can be linked. The second protocol (and security model) then also improves the users' privacy by hiding almost all information and also allows to establish different unlinkable identities, i.e., pseudonyms, with each service provider.

Roadmap. Section 3.1 contains the problem statement and additional preliminaries. The first ideal functionality $\mathcal{F}_{\mathsf{sso}}^1$ and the realizing protocol are presented in Section 3.2. The privacy-enhanced functionality $\mathcal{F}_{\mathsf{sso}}^2$ and the corresponding protocol are presented in Section 3.3. The required concrete zero-knowledge proofs for both protocols are given in Section 3.4, while the performance evaluations are presented in Section 3.5. Additional extensions are discussed in Section 3.6. This chapter is concluded in Section 3.7.

3.1 Introduction

Authenticating users is one of most important tasks in networking infrastructures. A widely deployed solution for this task is password-based single sign-on (SSO). Here, a user authenticates with a username and password towards a ticket-granting server that controls access for a certain trust domain. Upon successful password verification, a token (or "ticket") is issued to the user. The token then allows a user to authenticate towards service providers within that trust domain and a given time-frame. The main advantage of this approach is convenience for the users: they only need to remember a single username and password to login to numerous services. The most prominent and ubiquitous implementation of SSO is probably Kerberos [NT94].

However, virtually all currently deployed solutions in the real world have the same weakness – their entire security relies on the trustworthiness of the single ticket-granting server. If the ticket-granting server gets corrupted, it can impersonate the users towards all service providers and offline attack their passwords. In fact, given the low entropy of human-memorizable passwords, already a passive security breach that exposes the ticket-granting server's information to verify the passwords jeopardizes all user accounts [Gos12]. Typically, login passwords are verified against salted password hashes. Any compromise of these hashes and salt allows an adversary to brute-force the underlying passwords. In the context of SSO such a breach has a devastating impact, as the adversary immediately gains access to all of the users' accounts.

One possible mitigation strategy against these offline attacks is to rely on stronger user credentials, such as cryptographic keys or incorporating hardware tokens, but both comes with a significant loss in convenience for the users. Another approach is to avoid the single point of failure by distributing the user authentication and token generation across multiple ticket-granting servers. A number of solutions for that approach exist [CL12, CZLC05, JSS04, WYX13, ZLZL09]. However, they either focus on distributing only the token generation, but not the password check, or – for enhanced password security – require additional strong secrets on the user side. Thus, so far no solution exists that solely relies on passwords for the user authentication and is secure against offline attacks after a single ticket-granting server compromise.

3.1.1 Contribution

The question investigated in this chapter is whether password-based SSO can provide strong security and privacy guarantees. This is answered to the affirmative by formalizing and efficiently realizing UC-secure password-based distributed SSO. Therein, users are only required to remember a single password and username, and their passwords cannot be offline attacked unless all ticket-granting servers are corrupt. The same holds for impersonation attacks which an adversary can only perform when it corrupts all ticket-granting servers, or correctly guesses a user's password. Note, the latter, which is also known as online attack, is unavoidable as the authentication only relies on a password. To limit the power of such online attacks, the honest ticket-granting servers can – and should – refuse to participate in any further password checks for a user account when too many failed attempts have occurred. After all ticket-granting servers agree that a user has correctly authenticated itself, they issue the user a jointly computed authentication token. This token enables the user to access a plethora of services in the network, as long as the granted authentication token is valid.

To model SSO, the Universal Composition (UC) framework [Can01] is used, which has

many advantages when dealing with passwords [CLN15, CLNS16, CLN12, CHK+05b] (See also Chapter 4). For example, users often choose related or similar passwords in different contexts, and can make typos in their password attempts, e.g., trying to login with a slightly misspelled setup password. Moreover, the adversary may also be able to correctly guess passwords, e.g., if a user's password is too short, or if he was simply lucky. Using UC, all these inherent issues are naturally modeled. Further, strong security guarantees such as composability with arbitrary other protocols are given.

This chapter introduces two UC-functionalities $\mathcal{F}_{\mathsf{sso}}^1$ and $\mathcal{F}_{\mathsf{sso}}^2$. The functionality $\mathcal{F}_{\mathsf{sso}}^1$ is the basic version of the protocol, protecting the users' passwords and preventing impersonation attacks unless all ticket-granting servers are corrupted. It does not provide any additional privacy guarantees though, i.e., just as normal SSO it reveals the unique username towards service providers and all authentication sessions are fully linkable. The second functionality $\mathcal{F}_{\mathsf{sso}}^2$ then improves also the users' privacy by enabling them to be known under different identities, i.e., pseudonyms, towards different services. Both definitions allow to limit the validity of the generated authentication token by including time-stamps of the system. This required modeling the concept of time in the UC framework, which might be of independent interest.

Finally, the realizations for both functionalities are based on standard primitives, and are efficient enough for use in practice.

3.1.2 Related Work

The most prominent example of SSO is the already mentioned Kerberos [NT94, SNS88]. Because of its popularity, it was subject of formal treatment of the underlying primitives [BK11], yet also some stand-alone analysis of (variants of) the protocol [BCJ+11, BCJ+06, Wu99], and some attacks on related protocols were presented [BM91, Gro03, PW03]. This also includes an analysis of hardware-assisted, i.e., augmented versions of Kerberos [MPP14].

Another line of work focuses on symbolic analysis of SSO protocols [BJST08, CC08]. Clearly, there are a lot of other related functionalities, and their analysis, such as OpenID [RR06], SAML [ACC+08], or Shibboleth [MCC+04]. Moreover, there were also attempts to generalize certain concepts widely used in SSO protocols [PM03].

Distributing the token-generation across multiple ticket-granting servers is also not a new idea [CL12, CZLC05, JSS04, WYX13, ZLZL09]. However, they focus on distributing the token-generation, but not on distributing the password-check.

Also (but more remotely) related primitives are password-authenticated secret-sharing (PASS) schemes [BJSL11, CEN15, CLN12, CLLN14], password-authenticated key-exchange protocols (PAKE) [BPR00, BBC+13, CHK+05b, KMTG12, KM14] and credential-based key exchange schemes (CAKE) [CCGS10]. In particular, in a PASS scheme, a user can store a single value among multiple servers, protected by a password. This value can later be retrieved by a user, if the password entered was the same as used at registration. For SSO, however, this is not suitable, as the value stored does not change, and thus neither access rights nor token validity can be changed. Likewise, PAKE allows to establish a secure channel between two entities, if the password entered was correct. However, the password has to be entered each time, and thus orthogonal to SSO. Moreover, in CAKE a user needs to store its credential and therefore not stateless, which is exactly what is to be avoided in SSO.

3.1.3 Additional Preliminaries

For this chapter, some additional building blocks are required. These are presented next.

3.1.3.1 Semantically Secure (t, n)-Threshold Homomorphic Encryption Scheme

A (t, n)-threshold homomorphic encryption scheme TEnc is required. In a nutshell, a TEnc behaves as a standard encryption, with the exception that it supports a homomorphic property, and one needs $t + 1$ out of n partial secret keys to decrypt a ciphertext. In more detail, one can use the TEnc introduced by Camenisch et al. [CLLN14], as it perfectly fits the needs of the protocols. However, it is altered for the used notation. In particular, a label ℓ is added for the verifiable decryption shares. The construction is restated in Appendix A.

Definition 3.1 (Semantically Secure (t, n)-Threshold Homomorphic Public-Key Encryption Schemes)**.** *A semantically-secure (t, n)-threshold homomorphic public-key encryption scheme* TEnc *consists of the algorithms* $\{\mathsf{PPGen}_{\mathsf{TEnc}}, \mathsf{KeyGen}_{\mathsf{TEnc}}, \mathsf{Enc}_{\mathsf{TEnc}}, \mathsf{PDec}_{\mathsf{TEnc}}, \mathsf{VfDec}_{\mathsf{TEnc}}, \mathsf{Dec}_{\mathsf{TEnc}}, \mathsf{Hom}_{\mathsf{TEnc}}\}$*, such that:*

$\mathsf{PPGen}_{\mathsf{TEnc}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{TEnc}}$ *outputs the parameters* $\mathsf{pp}_{\mathsf{TEnc}}$ *of the scheme, where λ is the security parameter:*

$$\mathsf{pp}_{\mathsf{TEnc}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{TEnc}}(1^{\lambda})$$

It is assumed that $\mathsf{pp}_{\mathsf{TEnc}}$ *is implicit input to all other algorithms.*

$\mathsf{KeyGen}_{\mathsf{TEnc}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{TEnc}}$ *outputs the public key of the scheme and also a pair of partial secret and public keys:*

$$(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{1 \leq i \leq n}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TEnc}}(\mathsf{pp}_{\mathsf{TEnc}}, n, t)$$

$\mathsf{Enc}_{\mathsf{TEnc}}$**.** *The algorithm* $\mathsf{Enc}_{\mathsf{TEnc}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{TEnc}}$ *and the message $m \in \mathcal{MS}$ to encrypt. Note, the $\mathcal{MS}$ is required to be some cyclic multiplicatively written prime-order group $\mathbb{G}$. It outputs a ciphertext c:*

$$c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, m)$$

$\mathsf{PDec}_{\mathsf{TEnc}}$**.** *The algorithm* $\mathsf{PDec}_{\mathsf{TEnc}}$ *outputs a partial decryption share d_i, along with a proof π_i that the decryption was done correctly w.r.t. to* $\mathsf{pk}^i_{\mathsf{PTEnc}}$*, (or $\perp$, if the ciphertext is invalid) on input* $\mathsf{sk}^i_{\mathsf{PTEnc}}$*, label ℓ and a ciphertext c:*

$$(d_i, \pi_i) \xleftarrow{\$} \mathsf{PDec}_{\mathsf{TEnc}}(\mathsf{sk}^i_{\mathsf{PTEnc}}, c, \ell)$$

$\mathsf{VfDec}_{\mathsf{TEnc}}$**.** *The algorithm* $\mathsf{VfDec}_{\mathsf{TEnc}}$ *outputs a decision bit $d \in \{\mathsf{false}, \mathsf{true}\}$, verifying that a partial decryption share d_i of c is valid w.r.t.* $\mathsf{pk}^i_{\mathsf{PTEnc}}$*, ℓ and π_i:*

$$d \xleftarrow{\$} \mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}^i_{\mathsf{PTEnc}}, d_i, \pi_i, c, \ell)$$

Experiment $\mathsf{Indistinguishability}^{\mathsf{TEnc}}_{\mathcal{A},n,t}(\lambda)$:
$\mathsf{pp}_{\mathsf{TEnc}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{TEnc}}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{1\leq i\leq n}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TEnc}}(\mathsf{pp}_{\mathsf{TEnc}}, n, t)$
$b \xleftarrow{\$} \{0,1\}$
$(state_{\mathcal{A},1}, \mathcal{I}) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{pk}^i_{\mathsf{PTEnc}})_{1\leq i\leq n})$
If $|\mathcal{I}| > t$, return $a \xleftarrow{\$} \{0,1\}$
$((m^*_0, m^*_1), state_{\mathcal{A},2}) \xleftarrow{\$} \mathcal{A}(state_{\mathcal{A},1}, \mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}})_{i\in\mathcal{I}})$
If $m^*_0 \notin \mathcal{MS}^* \vee m_1 \notin \mathcal{MS}$:
  $c^* \leftarrow \perp$
Else:
  $c^* \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, m^*_b)$
$a \xleftarrow{\$} \mathcal{A}(state_{\mathcal{A},2}, c^*)$
return 1, if $a = b$
return 0

Figure 3.1: TEnc Indistinguishability

$\mathsf{Dec}_{\mathsf{TEnc}}$. *The algorithm* $\mathsf{Dec}_{\mathsf{TEnc}}$ *outputs the message* m*, on input of* $t+1$ *valid (and different) decryption shares* d_i*,* $\mathsf{pk}_{\mathsf{TEnc}}$ *and* c*:*

$$m \xleftarrow{\$} \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_i)_{i\in[1,t+1]}, c)$$

$\mathsf{Hom}_{\mathsf{TEnc}}$. *The deterministic algorithm* $\mathsf{Hom}_{\mathsf{TEnc}}$ *outputs a ciphertext* c'*, on input of two ciphertexts* c_1*,* c_2*, and the public key* $\mathsf{pk}_{\mathsf{TEnc}}$*:*

$$c' \leftarrow \mathsf{Hom}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, c_1, c_2)$$

Correctness. For each TEnc, the usual correctness properties must hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{TEnc}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{TEnc}}(1^\lambda)$, for all $n \in \mathbb{N}^+$, for all $t \in \{s \mid s < n, s \in \mathbb{N}^0\}$, for all $(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{i\in[1,n]}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TEnc}}(\mathsf{pp}_{\mathsf{TEnc}}, n, t)$, for all $m \in \mathcal{MS}$, for all $\ell \in \{0,1\}^*$, and for all $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{ENC}}, m)$, for all $(d_i, \pi_i) \xleftarrow{\$} \mathsf{PDec}_{\mathsf{TEnc}}(\mathsf{sk}^i_{\mathsf{PTEnc}}, c, \ell)$, it holds that $\mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}^i_{\mathsf{PTEnc}}, d_i, \pi_i, c, \ell) = \mathsf{true}$ and for all subsets $S \subseteq \{d_i\}_{i\in[1,n]}$ of cardinality $t+1$, it holds that $m = \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, S, c)$, yet also for all c_1, c_2, with $c' \leftarrow c_1 \odot c_2$, it holds that the decryption of c' yields the product of the plaintexts of c_1 and c_2. Once more, this definition captures perfect correctness.

Also some security guarantees are required, introduced next.

Indistinguishability. Roughly, this property guarantees that an adversary holding only t partial secret keys does not learn anything about a plaintext contained in a ciphertext, even if it can choose the corresponding plaintexts.

Definition 3.2 (Indistinguishability)**.** *An encryption scheme* TEnc *is indistinguishable, if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that for all polynomial* $t < n \in \mathbb{N}^+$*:*

$$\left|\Pr[\mathsf{Indistinguishability}^{\mathsf{TEnc}}_{\mathcal{A},n,t}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 3.1.

3.1.3.2 Zero-Knowledge Non-Interactive Proof of Knowledge Systems

Let L be an NP-language with associated witness relation R, i.e., such that $L = \{x \mid \exists w : R(x, w) = \mathsf{true}\}$. In a nutshell, a non-interactive zero-knowledge proof of knowledge allows to verify that the generator of such a proof knows a witness w for some statement x without revealing that witness. More formally, such a system is defined as follows.

Definition 3.3. *A zero-knowledge non-interactive proof of knowledge system* $\mathsf{NIZKPoK}$ *consists of three algorithms* $\{\mathsf{PPGen}_{\mathsf{NIZKPoK}}, \mathsf{Prove}_{\mathsf{NIZKPoK}}, \mathsf{Verify}_{\mathsf{NIZKPoK}}\}$*, such that:*

$\mathsf{PPGen}_{\mathsf{NIZKPoK}}$**.** *The algorithm* $\mathsf{crs}_{\mathsf{NIZKPoK}}$ *outputs public parameters of the scheme, where* λ *is the security parameter:*

$$\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda, L)$$

For simplicity, it assumed that $\mathsf{crs}_{\mathsf{NIZKPoK}}$ *is an implicit input to all other algorithms, while the language* L *is clear from the context.*

$\mathsf{Prove}_{\mathsf{NIZKPoK}}$**.** *The algorithm* $\mathsf{Prove}_{\mathsf{NIZKPoK}}$ *outputs the proof* π*, on input of the statement* x *to be proven, and the corresponding witness* w*:*

$$\pi \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}(x, w)$$

$\mathsf{Verify}_{\mathsf{NIZKPoK}}$**.** *The deterministic algorithm* $\mathsf{Verify}_{\mathsf{NIZKPoK}}$ *verifies the proof* π*, w.r.t. to some statement* x*, where* $d \in \{\mathsf{false}, \mathsf{true}\}$*:*

$$d \leftarrow \mathsf{Verify}_{\mathsf{NIZKPoK}}(x, \pi)$$

For the sake of readability, a somewhat informal CS-notation is used, derived from Camenisch and Stadler [CS97]. For example, the notation $\pi \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(g_1) : C = \mathsf{Enc}_{\mathsf{TEnc}}(g_1)\}(\ell)$ denotes the computation of a non-interactive, simulation-sound extractable, zero-knowledge proof of knowledge of the plaintext g_1 contained in C (which is assumed to be public), with an attached label ℓ. Sometimes only "verify π" is used for verification of a proof π. It is assumed that the public parameters and the statement to be proven, are also input to the proof system and public. This is not make explicit to increase readability. The detailed and formal, NIZK-proofs with the instantiations proposed are given in Section 3.4.

Experiment $\mathsf{Zero\text{-}Knowledge}^{\mathsf{NIZKPoK}}_{\mathcal{A},\mathsf{SIM},L}(\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau) \xleftarrow{\$} \mathsf{SIM}_1(1^\lambda, L)$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{P_b(\cdot,\cdot)}}(\mathsf{crs}_{\mathsf{NIZKPoK}})$
where oracle P_0 on input x and w:
return $\pi \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}(x, w)$, if $R(x, w) = \mathsf{true}$
return $\perp$
and oracle P_1 on input (x, w):
return $\pi \xleftarrow{\$} \mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, x)$, if $R(x, w) = \mathsf{true}$
return $\perp$
return 1, if $a = b$
return 0

Figure 3.2: NIZKPoK Zero-Knowledge

Correctness. In the context of (zero-knowledge) proof-systems, correctness is sometimes also referred to as completeness. More precisely, it is required that for all $\lambda \in \mathbb{N}$, for all "suitable" L, for all $\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda, L)$, for all $x \in L$, for all w such that $R(x, w) = \mathsf{true}$, for all $\pi \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}(x, w)$, it must hold that $\mathsf{Verify}_{\mathsf{NIZKPoK}}(\mathsf{crs}_{\mathsf{NIZKPoK}}, x, \pi) = \mathsf{true}$. As usual, some correctness and security notions are required, which are introduced next. These are mainly taken from Groth [Gro06].

Two different security notions are required, i.e., zero-knowledge and simulation-sound extractability.

Zero-Knowledge. In a nutshell, zero-knowledge says that the receiver of the proof π does not learn anything except the validity of the statement. It is assumed that the distribution of $\mathsf{crs}_{\mathsf{NIZKPoK}}$ output by SIM_1 is distributed identically to $\mathsf{PPGen}_{\mathsf{NIZKPoK}}$.

Definition 3.4 (Zero-Knowledge)**.** *A non-interactive proof system* NIZKPoK *is zero-knowledge, if for a fixed language L, for any PPT adversary $\mathcal{A}$, there exists an PPT simulator $\mathsf{SIM} = (\mathsf{SIM}_1, \mathsf{SIM}_2)$ such that there exists a negligible function ν such that:*

$$\left|\Pr[\mathsf{Zero\text{-}Knowledge}^{\mathsf{Prove}_{\mathsf{NIZKPoK}}}_{\mathcal{A},\mathsf{SIM},L}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 3.2. Here, τ is the trapdoor for the simulation.

Simulation-Sound Extractability. This security notion says that an adversary cannot generate a proof π^* for a statement it does not know a witness for, while the proof-system is also of knowledge, i.e., the witness w can be extracted from any non-simulated proof π. Clearly, this also implies that the proof-system is non-malleable. Note, the simulation trapdoor τ is given to the adversary, i.e., the adversary can simulate proofs itself.

Experiment $\mathsf{SimSoundExt}^{\mathsf{NIZKPoK}}_{\mathcal{A},E,L}(\lambda)$
$(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \xi) \xleftarrow{\$} E_1(1^\lambda, L)$
$(x^*, \pi^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{SIM}(\cdot)}}(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau)$
$\mathcal{Q} \leftarrow \emptyset$
where oracle SIM on input x:
obtain $\pi \xleftarrow{\$} E_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, x)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(x, \pi)\}$
return π
$w^* \xleftarrow{\$} E_3(\mathsf{crs}_{\mathsf{NIZKPoK}}, \xi, x, \pi)$
return 1, if $\mathsf{Verify}_{\mathsf{NIZKPoK}}(x^*, \pi^*) = \mathsf{true} \ \wedge \ R(x^*, w^*) = \mathsf{false} \ \wedge \ (x^*, \pi^*) \notin \mathcal{Q}$
return 0

Figure 3.3: NIZKPoK Simulation Sound Extractability

Definition 3.5 (Simulation-Sound Extractability)**.** *A zero-knowledge non-interactive proof system* NIZKPoK *is said to be simulation-sound extractable, if for a fixed language L, for any PPT adversary $\mathcal{A}$, there exists a PPT extractor/simulator $E = (E_1, E_2, E_3)$, such that there exists a negligible function ν such that:*

$$\Pr[\mathsf{SimSoundExt}^{\mathsf{NIZKPoK}}_{\mathcal{A},E,L}(\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 3.3.

Note, E_1, E_2 are required to behave exactly as $(\mathsf{SIM}_1, \mathsf{SIM}_2)$ from the zero-knowledge definition [Gro06], while ξ is the extraction trapdoor.

Definition 3.6. *A* $\mathsf{Prove}_{\mathsf{NIZKPoK}}$ *is said to be secure, if it is complete, simulation-sound extractable and zero-knowledge, as defined by Groth [Gro06].*

The language L will no longer be made explicit, as it can be derived from the construction what L is exactly. It is also required that an arbitrary label $\ell \in \{0, 1\}$ can be attached to the proof, which also contains the public parameters and the statement to be proven, i.e., can be transformed into a "signature of knowledge" [CL06, FO11]. This is essentially used to avoid some potential problems, depending on the used proof system, discovered by Bernard et al. [BPW12a]. See Section 3.4 for an additional discussion.

Note, as the proofs are non-malleable and non-interactive, it is already implied that they are secure in a concurrent setting [SCO+01], while simulation-sound extractability implies non-malleability [SCO+01]. Prohibiting re-usage in different contexts can be achieved by appending context-information [CLLN14], e.g., the *sid*.

To improve performance, it is shown in Section 3.4 how to use more efficient primitives. Moreover, in the proposed instantiation, not all witnesses need to be extractable by the simulator, but only some, which improves performance [CKS11]. To also account for this, the witnesses which need to be extractable are denoted as $\overline{m}$. In fact, this optimization is directly used in the concrete proofs presented in Section 3.4.

3.1.3.3 Secret Sharing Schemes

The second protocol requires a perfectly secure $(n, n+1)$-secret-sharing scheme, e.g., the one presented by Shamir [Sha79]. Here, a party can share a value across n participants, while all n shares are required to re-construct the secret.

Definition 3.7 (Secret Sharing Scheme)**.** *A secret-sharing scheme* SS *consists of the algorithms* $\{\mathsf{PPGen}_{\mathsf{SS}}, \mathsf{Share}_{\mathsf{SS}}, \mathsf{Combine}_{\mathsf{SS}}\}$ *such that:*

$\mathsf{PPGen}_{\mathsf{SS}}$**.** *This algorithm outputs parameters:*

$$\mathsf{pp}_{\mathsf{SS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SS}}(1^\lambda, n)$$

The public parameters are assumed to be input to all following algorithms.

$\mathsf{Share}_{\mathsf{SS}}$**.** *To share a secret* m*, do:*

$$\{s_1, s_2, \ldots, s_n\} \xleftarrow{\$} \mathsf{Share}_{\mathsf{SS}}(m)$$

$\mathsf{Combine}_{\mathsf{SS}}$**.** *To (deterministically) reconstruct a secret, do:*

$$m \leftarrow \mathsf{Combine}_{\mathsf{SS}}(\{s_1, s_2, \ldots, s_n\})$$

Correctness. For each SS, the usual correctness properties must hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $n \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{SS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SS}}(1^\lambda, n)$, for all $m \in \mathcal{MS}$, for all $\{s_1, s_2, \ldots, s_n\} \xleftarrow{\$} \mathsf{Share}_{\mathsf{SS}}(m)$, it holds that $m = \mathsf{Combine}_{\mathsf{SS}}(\{s_1, s_2, \ldots, s_n\})$.

Perfect Privacy. It is required that $n-1$ shares do leak nothing (in the information-theoretical) sense. The following definition is derived from Beimel [Bei11]. Note, the definition captures *perfect* privacy, as the probability is also taken over all random coins used by the challenger.

Definition 3.8 (Perfect Privacy)**.** *A secret-sharing scheme* SS *is perfectly private, if for all* $n \in N$*, and for all unbounded* $\mathcal{A}$*, the following holds:*

$$\Pr[\mathsf{PerfectPrivacy}^{\mathsf{SS}}_{\mathcal{A},\mathsf{n}}(1^\lambda) = 1] - \tfrac{1}{2} = 0$$

The corresponding experiment is depicted in Figure 3.4.

Definition 3.9. *A secret-sharing scheme* SS *is said to be secure, if it is correct, and perfectly private.*

Experiment $\mathsf{PerfectPrivacy}^{\mathsf{SS}}_{\mathcal{A},n}(\lambda)$:
$\mathsf{pp}_{\mathsf{SS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SS}}(1^\lambda, n)$
$b \xleftarrow{\$} \{0,1\}$
$m_1 \xleftarrow{\$} \mathcal{MS}$
$m_2 \xleftarrow{\$} \mathcal{MS}$
$r \xleftarrow{\$} [1,n]$
$\{s_1^{m_1}, s_2^{m_1}, \ldots, s_n^{m_1}\} \xleftarrow{\$} \mathsf{Share}_{\mathsf{SS}}(m_1)$
$\{s_1^{m_2}, s_2^{m_2}, \ldots, s_n^{m_2}\} \xleftarrow{\$} \mathsf{Share}_{\mathsf{SS}}(m_2)$
If $b = 0$, let $T \leftarrow \{s_1^{m_1}, s_2^{m_1}, \ldots, s_n^{m_1}\} \setminus \{s_r^{m_1}\}$
If $b = 1$, let $T \leftarrow \{s_1^{m_2}, s_2^{m_2}, \ldots, s_n^{m_2}\} \setminus \{s_r^{m_2}\}$
$a \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\mathsf{SS}}, m_1, m_2, T)$
return 1, if $a = b$
return 0

Figure 3.4: SS Perfect Privacy

3.1.3.4 Signatures with Efficient Zero-Knowledge Proofs

The second protocol also requires that one can easily prove knowledge of a signature σ w.r.t. to some public key and a message value $m = (m_1, m_2, \ldots, m_n)$, where even some blocks m_i may be hidden from the verifier. In terms of the (more or less informal) Camenisch-Stadler (CS) notation [CS97], this is expressed as:

$$\pi \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\sigma, m_1) : \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}, m, \sigma) = 1\}$$

In more detail, in this example, m_1 is hidden, while m_2 to m_n are public. A suitable instantiation of such a signature scheme are CL-signatures [CL02b], the scheme by Abe et al. [AGHO11] or the scheme by Groth [Gro15]. See Section 3.4 for the formal proofs with the signature scheme instantiated with the scheme by Groth [Gro15]. Note, however, that this signature scheme requires additional public parameters. See Appendix B for a discussion.

3.1.3.5 Timestamps and Univeral Composition

As the authentication tokens must only be valid for a certain amount of time, the functionalities need to incorporate some concept of "time". This is, in both functionalities, modeled in such a way that any party can advance the clock, while all other parties can receive the current time *time*. The time is, moreover, modeled as an abstract unit, which only property is that it continues forward. This models synchronized clocks in the network, without the need to explicitly define what "time" really is.

To circumvent the requirement of synchronized clocks, one can alter the ideal functionalities in such a way that each party $\mathcal{P}_i$ has its own clock $time_i$, i.e., it is local, which can be advanced by the environment. All the computations and checks are then performed w.r.t. to the local $time_i$, and not the global *time*. However, to make the functionalities, proofs, and protocols more readable, it was chosen to use the "simpler" definition.

Canetti et al. also define a global functionality, which can be used in this context [CHMV17].

3.2 Basic Distributed UC-Secure Single Sign-On

In this section, the functionality $\mathcal{F}_{\mathsf{sso}}^{1}$ is presented. It offers the basic functionality one expects from a distributed SSO protocol. A user sets up an account and can later obtain an authentication token with which it can access service providers. The ticket-granting servers do not learn anything about mistyped password attempts at token generation (which is also true for the service providers which are not involved at all), while all ticket-granting servers need to be corrupt to learn the password used at registration, as this is clearly the most basic intention behind a distributed SSO functionality.

An extended functionality $\mathcal{F}_{\mathsf{sso}}^{2}$ is presented in Section 3.3, which offers additional privacy features, while the realizing protocol is slightly less efficient.

In more detail, both $\mathcal{F}_{\mathsf{sso}}^{1}$ and $\mathcal{F}_{\mathsf{sso}}^{2}$, have the following participants: a set of *ticket-granting servers* T, and a set of *service providers* S, which accept authentication tokens by the *users*. For brevity, it is assumed that there is a canonical ordering of each set of ticket-granting servers T and service providers S, e.g., using IP-addresses or DNS-entries.

At the very beginning of the protocol, all servers, i.e., the ticket-granting servers T and the service providers S, need to setup their public keys, while also registering them at a PKI, modeled as $\mathcal{F}_{\mathsf{CA}}$, i.e., they are authenticated. Then, a non-authenticated user $\mathcal{U}$ can register an account on the ticket-granting servers T with its password pwd and some username uid.

To generate a token (which does not exist as a cryptographic value in the ideal world), another party enters a password attempt pwd', while the ticket-granting servers start a distributed protocol to check whether the password attempt matches the password used at registration. The ticket-granting servers can then decide if they want to continue the request, i.e., at this point the throttling mechanism kicks in. If they continue and if the password attempt was correct, they also input the set of service providers a user can access and how long the token remains valid. It is important to notice that the user initiating the token generation may not be the same as the user which has registered the account. This models the case where a user either mistypes its username uid, or simply uses a different terminal, e.g., a PC or laptop within a huge company, while users do not need to store any state between protocol runs.

After successful token generation, a user can access a service provider using the obtained token. In the given functionalities, only a user can send a message to the service provider. The step to obtain a bi-directional channel is easy: the channel can be boot-strapped by sending a symmetric key k which is then used in some upper layer protocol to communicate in an authenticated and private way using, e.g., AES and MACs.

3.2.1 The Ideal Functionality $\mathcal{F}_{\mathsf{sso}}^{1}$

For simplicity, it is required that $sid = (\mathsf{T}, \mathsf{S}, sid')$, where the users are not part of the sid to avoid implying some sort of authenticated channels, as the user $\mathcal{U}$ should only need to remember its username uid, the sid, and its password pwd. Thus, only the service providers S, and the ticket-granting servers T, have authenticated public keys, which is a reasonable assumption. It is also assumed that the query identifiers qid are globally unique, and thus omit checking for

1. `Update Timestamp.` On input $(\mathsf{UPDATETIME}, sid, time')$ from any party $\mathcal{P}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time' \notin \mathbb{N}$, ignore.
   - If $time' > time$, set $time \leftarrow time'$.
   - Output $(\mathsf{UPDATETIME}, sid, time)$ to $\mathcal{P}$.
2. `Request Timestamp.` On input $(\mathsf{REQUESTTIME}, sid)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, ignore.
   - Output $(\mathsf{TIME}, sid, time)$ to $\mathcal{P}$.
3. `Setup Service Provider.` On input $(\mathsf{SETUPSERVICE}, sid)$ from service provider $\mathcal{S}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $time = -1$, or $\mathcal{S} \notin \mathsf{S}$, ignore.
   - If there is a record $(sid, \mathsf{ssetup\text{-}rec}, \mathcal{S})$, ignore.
   - Create record $(sid, \mathsf{ssetup\text{-}rec}, \mathcal{S})$.
   - Send $(\mathsf{SETUPSERVICE}, sid, \mathcal{S})$ to adversary $\mathcal{A}$.
4. `Setup Ticket-Granting Server.` On input $(\mathsf{SETUPSERVER}, sid)$ from ticket server $\mathcal{T}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $time = -1$, or $\mathcal{T} \notin \mathsf{T}$, ignore.
   - If there is a record $(sid, \mathsf{tsetup\text{-}rec}, \mathcal{T})$, ignore.
   - Create record $(sid, \mathsf{tsetup\text{-}rec}, \mathcal{T})$.
   - Send $(\mathsf{SETUPSERVER}, sid, \mathcal{T})$ to adversary $\mathcal{A}$.

Figure 3.5: Initialization and clock interfaces for $\mathcal{F}_{\mathsf{sso}}^{1}$. The variable *time* is set to -1 at the beginning.

duplicates in the functionality. This can, e.g., be achieved by exchanging nonces between all parties beforehand, and then concatenating them [BLR04]. If this method is used, it is assumed that the parties ignore all input for which they did not contribute randomness.

The functionality is split into several parts to increase readability. Namely, there are the setup interfaces (Figure 3.5), the registration interfaces (Figure 3.6), the authentication token generation interfaces (Figure 3.7 and 3.8) and the communication interfaces for accessing the service providers (Figure 3.9).

Note, in the description of the functionality "ignore" means that it gives control back to the caller with an implicit error message. Moreover, if the functionality outputs a value to some party, control is given to that party. Hence, the functionality directly gives up control.

The Setup Interfaces. The setup interfaces, depicted in Figure 3.5, allow parties to setup their state. This may, e.g., be key pairs. As already mentioned, these interfaces are meant for the ticket-granting servers and the service providers, as they are authenticated. This group also contains the interfaces to advance the clock by the environment and an interface to receive the current time. In more detail, there are the following interfaces:

1. The $\mathsf{UPDATETIME}$ interface allows a party $\mathcal{P}$ to advance the current time *time*. Note, the clock can only be advanced, as time "obviously" only has one direction in the real world, ignoring some physical thoughts [Vel12].
2. The $\mathsf{REQUESTTIME}$ interface allows any party to obtain the current time. As this models a local interaction, the current time is directly returned, without the adversary intervening.
3. The $\mathsf{SETUPSERVICE}$ interface allows a service provider to initialize itself, i.e., to generate the public keys, and to register them with some PKI.

5. `Create Account Request.` On input (CREATEACCREQ, *sid*, *qid*, uid, pwd) from party $\mathcal{U}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$ or $time = -1$, or a record (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, $\cdot$, $\cdot$) exists, ignore.
   - Create record (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, pwd, false).
   - If all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, send (CREATEACCREQ, *sid*, *qid*, $\mathcal{U}$, uid, pwd) to $\mathcal{A}$.
   - Send (CREATEACCREQ, *sid*, *qid*, $\mathcal{U}$, uid) to $\mathcal{A}$.
6. `Account Created.` On input (ACCCREATED, *sid*, *qid*, $\mathcal{U}$, uid) from adversary $\mathcal{A}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
   - If a record (user-rec, *sid*, $\cdot$, $\cdot$, uid, $\cdot$, true) exists and not all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, ignore.
   - If a record (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, pwd, false) exists for some pwd, update it to (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, pwd, true), otherwise ignore.
   - Output (ACCCREATED, *sid*, *qid*, uid) to $\mathcal{U}$ (delayed).

Figure 3.6: Registration interfaces for $\mathcal{F}_{\mathsf{sso}}^{1}$

4. The SETUPSERVER interface allows a ticket-granting server to initialize itself. As for the SETUPSERVICE interface, this means that the ticket-granting server can generate its key pairs, and register them with a PKI.

The Registration Interfaces. The registration interfaces, depicted in Figure 3.6, allow users to register an account with the given ticket-granting servers T.

5. The CREATEACCREQ interface allows a user $\mathcal{U}$ to register a new account in the system with user id uid. The user $\mathcal{U}$ also needs to provide a password pwd, which it can later use to authenticate towards the ticket-granting servers. The password pwd used is only learned, if all ticket-granting servers are corrupt at the same time. In the case where the adversary "overwrites" such a setup request, the adversary provides the password, while in the case of all corrupt servers more than one account per uid may be registered, as the users do not communicate directly, and thus cannot know whether there are other accounts with the same uid.
6. The ACCCREATED interface allows the adversary to activate an account. Note, the account may already be created on the servers, while the user never received an output, e.g., if the adversary holds back the acknowledgments from the ticket-granting servers.
   Thus, the output to the user is delayed. Moreover, all honest servers need to agree on the same login data, if at least one ticket-granting server is honest, i.e., only a single account can be activated in this case.

The Token Generation Interfaces. The token generation interfaces, depicted in Figure 3.7, allow users to obtain an authentication token, which authenticates users to the service providers. The password attempt pwd′ provided has to be the same as used during the registration, as long as at least one ticket-granting server remains honest.

7. The RECVTOKENREQ interface allows a user $\mathcal{U}$ to try to obtain an authentication token. It needs to provide a password attempt pwd′ and some username uid. Note, the adversary may still overrule token-generation requests with its own password attempt, as the users are not authenticated. However, any further requests with the *qid* overwritten are then ignored.

7. `Create Token Generation Request.` On input $(\mathsf{RECVTOKENREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd}')$ from party $\mathcal{U}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
   - Create record $(\mathsf{tokenreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathsf{pwd}')$.
   - Send $(\mathsf{RECVTOKENREQ}, sid, qid, \mathcal{U}, \mathsf{uid})$ to $\mathcal{A}$.
8. `Get Token Generation Permission.` On input $(\mathsf{TKNRECVPRM}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T})$ from adversary $\mathcal{A}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{T} \notin \mathsf{T}$, or $time = -1$, ignore.
   - If there is record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{A}, \cdot, \mathcal{T}, \cdot, \cdot)$, ignore.
   - If there is no record $(\mathsf{user\text{-}rec}, sid, \cdot, \cdot, \mathsf{uid}, \cdot, \mathsf{true})$, ignore.
   - If there is no record $(sid, \mathsf{tsetup\text{-}rec}, \mathcal{T})$, ignore.
   - If there is a record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \cdot, \cdot)$, ignore.
   - Create record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \perp_{time}, \perp_{\mathsf{S}})$.
   - Output $(\mathsf{TKNRECVPRM}, sid, qid, \mathsf{uid})$ to $\mathcal{T}$.
9. `Token Generation Request Granted.` On input $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$ from party $\mathcal{T}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{T} \notin \mathsf{T}$, $\mathsf{S}' \not\subseteq \mathsf{S}$, $time = -1$, or $time' \notin \mathbb{N}$, ignore.
   - If there is no record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \cdot, \mathsf{uid}, \mathcal{T}, \perp_{time}, \perp_{\mathsf{S}})$, ignore.
   - Update the record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \cdot, \cdot)$ to $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, time', \mathsf{S}')$.
   - Send $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, \mathcal{T})$ to $\mathcal{A}$.
10. `Test Password.` On input $(\mathsf{TESTPW}, sid, qid, \mathsf{pwd}')$ from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
    - If there is a record $(\mathsf{testpw\text{-}rec}, sid, qid)$, ignore.
    - Create record $(\mathsf{testpw\text{-}rec}, sid, qid)$.
    - If not all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, ignore, if not for all honest $\mathcal{T}_i \in \mathsf{T}$ there exists a record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, time', \mathsf{S}')$.
    - If not all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, let $b \leftarrow \mathsf{true}$, if there exists a record $(\mathsf{user\text{-}rec}, sid, \cdot, \cdot, \mathsf{uid}, \mathsf{pwd}'', \mathsf{true})$, pwd'' taken from record $(\mathsf{tokenreq\text{-}rec}, sid, qid, \cdot, \cdot, \mathsf{pwd}'')$, and $b \leftarrow \mathsf{false}$ otherwise. Send $(\mathsf{TESTPW}, sid, qid, b)$ to $\mathcal{A}$.
    - Ignore, if not all $\mathcal{T}_i \in \mathsf{T}$ are corrupt.
    - Send $(\mathsf{TESTPW}, sid, qid, \mathsf{true})$ to $\mathcal{A}$, if there is a record $(\mathsf{tokenreq\text{-}rec}, sid, qid, \mathcal{U}, \cdot, \mathsf{pwd}')$, and $(\mathsf{TESTPW}, sid, qid, \mathsf{false})$ otherwise.

Figure 3.7: First set of token generation interfaces for $\mathcal{F}_{\mathsf{sso}}^{1}$

8. The TKNRECVPRM interface allows the adversary to trigger a specific ticket-granting server $\mathcal{T}$ to ask for permission to proceed checking a password, i.e., this is the hook to the external throttling mechanism. This interface can only be called, if there is actually an ongoing request, while the account with uid must be activated and the ticket-granting must have been set up.
9. The TOKENGENGRANT interface allows the environment to decide whether the ticket-granting server $\mathcal{T}$ is allowed to proceed with checking the password. Note, this interface only allows whether a ticket-granting server is allowed to continue the password check, i.e., it does not yet learn whether the password attempt was correct or not.

   Moreover, a ticket-granting server also decides which service providers a given user uid can

11. `Notify Server.` On input $(\mathsf{NOTIFY}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T})$ from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{T} \notin \mathsf{T}$, or $time = -1$, ignore.
    - If not for all honest $\mathcal{T}_i \in \mathsf{T}$ there exists a record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, time', \mathcal{S}')$, where $time' \neq \bot_{time}$, and $\mathcal{S}' \neq \bot_{\mathsf{S}}$, ignore.
    - Ignore, if there is a record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \cdot)$.
    - Let $b \leftarrow \mathsf{true}$, if there is a record $(\mathsf{tokenreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathsf{pwd}, \mathsf{true})$, and a record $(\mathsf{user\text{-}rec}, sid, \cdot, \cdot, \mathsf{uid}, \mathsf{pwd}, \mathsf{true})$. Set $b \leftarrow \mathsf{false}$ otherwise.
    - Create record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, b)$.
    - Output $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, b)$ to $\mathcal{T}$.
12. `Continue Notify.` On input $(\mathsf{CONTNOTIFY}, sid, qid, \mathsf{uid})$ from party $\mathcal{T}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{T} \notin \mathsf{T}$, or $time = -1$, ignore.
    - If there is no record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \mathsf{true})$, ignore.
    - If there is a record $(\mathsf{srvnot\text{-}rec}, sid, qid)$, ignore.
    - Create record $(\mathsf{srvnot\text{-}rec}, sid, qid)$.
    - Send $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathcal{T}, time', \mathsf{S})$ to $\mathcal{A}$, where $time'$, and S, are taken from record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, time', \mathcal{S}')$.
13. `Token Generation Complete.` On input $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathcal{U}, \mathsf{uid}, time'', \mathsf{S}'')$ from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
    - If all $\mathcal{T}_i \in \mathsf{T}$ are honest, let $\mathsf{S}'' \leftarrow \mathsf{S}$ and $time'' \leftarrow \infty$.
    - If there is a record of the form $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$, ignore.
    - If not for all honest $\mathcal{T}_i \in \mathsf{T}$ there exists a record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathsf{uid}, \mathcal{T}_i, \mathsf{true})$, ignore.
    - For all existing records of the form $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}_i, time_i, \mathsf{S}_i)$, let $time' \leftarrow \min(\{time_i\} \cup time'')$, and $\mathsf{S}' \leftarrow \bigcap \mathsf{S}_i \cap \mathsf{S}''$.
    - Create record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, time', \mathsf{S}', (time_i, \mathsf{S}_i)_{i\in[1,n]})$.
    - Output $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$ to $\mathcal{U}$.

Figure 3.8: Second set of token generation interfaces for $\mathcal{F}_{\mathsf{sso}}^{1}$

access and how long the token is valid once the password check was successful.

10. The TESTPW interface allows the adversary to learn whether the password attempt was correct, but only if all honest ticket-granting servers agree that they participate. In the case that all ticket-granting servers are corrupt, the adversary moreover receives a free guess on the used password.

    This is not avoidable, as users do not store any state, and thus a user has to trust the values obtained from the ticket-granting servers. Note, however, that this interface has no real-world pendant, as its only purpose is to give the adversary enough information for the simulation.
11. The NOTIFY interface allows the adversary to decide whether a ticket-granting server $\mathcal{T}$ learns if the used password was correct. This interface can only be called, if all honest servers agree to continue the password check.
12. The CONTNOTIFY interface allows a ticket-granting server to continue the token generation request. This interface is necessary, as due to the prior interface the environment has control, i.e., the throttling mechanism needs to acknowledge that it has received the information whether the passwords matched.
13. The TKNRECVCOMPL interface allows the adversary to decide when the user $\mathcal{U}$ obtains the

14. `Send Message.` On input $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, m, \mathcal{S}, \Delta)$ from party $\mathcal{U}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{S} \notin \mathsf{S}$, $time = -1$, or $\Delta \notin \mathbb{N}$, ignore.
    - If there is a record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{U}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$, $\mathcal{U} \neq \mathcal{A}$, ignore.
    - If there is no record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid', \mathcal{U}, \mathsf{uid}, time', \mathsf{S}', time'', \mathsf{S}'')$, where $\mathcal{S} \in \mathsf{S}'$, and $time + \Delta \leq time'$, ignore, if not all $\mathcal{T}_i \in \mathsf{T}$ and $\mathcal{U}$ are corrupt.
    - If all $\mathcal{T}_i \in \mathsf{T}$ and $\mathcal{U}$ are corrupt, set $time'' \leftarrow \infty$, and $\mathsf{S}'' \leftarrow \mathsf{S}$.
    - Create record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time + \Delta, \mathsf{false})$.
    - If $\mathcal{S}$ is corrupt, send $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, \mathcal{U}, \mathcal{S}, \mathsf{S}', time'', \mathsf{S}'', m, \Delta)$ to $\mathcal{A}$.
    - Send $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, \mathcal{U}, \mathcal{S}, \mathsf{S}', time'', \mathsf{S}'', \Delta)$ to $\mathcal{A}$.
15. `Receive Message.` On input $(\mathsf{RECVMESSAGE}, sid, qid)$ from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
    - If there is a record $(\mathsf{msg\text{-}rec}, sid, qid, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{true})$, ignore.
    - If there is no record $(\mathsf{msg\text{-}rec}, sid, qid, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{false})$, ignore.
    - Update the record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{false})$ to $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{A}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{true})$, where $\mathcal{U} \neq \mathcal{A}$, if it exists.
    - If $time > time'$, ignore, if there does not exist a record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{A}, \mathsf{uid}, m, \mathcal{S}, n, \mathsf{S}'', \cdot, \mathsf{false})$, where $n \geq time$.
    - Update record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{A}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{false})$ to $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{A}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{true})$, if it exists.
    - Output $(\mathsf{RECVMESSAGE}, sid, qid, \mathsf{uid}, m)$ to $\mathcal{S}$, where uid and m are taken from record $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{A}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{true})$ if such a record and exists and from $(\mathsf{msg\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, m, \mathcal{S}, time'', \mathsf{S}'', time', \mathsf{true})$, where $\mathcal{U} \neq \mathcal{A}$, otherwise.

Figure 3.9: Communication interfaces for $\mathcal{F}_{\mathsf{sso}}^{1}$

token. This interface enforces that every ticket-granting server agrees to continue the request, but also that each honest ticket-granting server has learned whether the password attempt was correct.

Clearly, the honest ticket-granting servers only proceed, if the password entered was correct. Moreover, the adversary now has the possibility to overwrite the access rights of a user in the name of any corrupt server. Still, the adversary can only "downgrade" a user's access rights, as long as at least one ticket-granting server remains honest, i.e., the adversary cannot interfere in this case.

Next the communication interfaces, depicted in Figure 3.9, are explained in greater detail. These interfaces allow a user $\mathcal{U}$ to send a message to a service.

14. The SENDMESSAGE interface allows a user $\mathcal{U}$ to send a message m to some service provider $\mathcal{S}$. Here, a user $\mathcal{U}$ also specifies a Δ, specifying how long a message should be accepted by the service provider. Moreover, this interface only continues, if actually a valid token was generated.
15. The RECVMESSAGE interface allows the adversary to decide that a specific message is received by a service provider. The output is only triggered, if a valid token for the user $\mathcal{U}$ exists, and the current *time* is not greater than what the user specified, and a user actually sent the message in question. Note, however, that a corrupt user may also have a valid token which it can use to overwrite the message send. As old tokens remain valid by definition, this

cannot be avoided. Thus, an adversarially generated message always "overrules" a genuine message, as only one message per *qid* is accepted.

Finally, for all protocols presented, it is assumed that the message encrypted has a maximum length to avoid additional leakage.

3.2.2 Realizing Protocol for $\mathcal{F}_{\mathsf{sso}}^{1}$

As for the ideal functionality, the protocol presented is also split into four parts. Namely, setup, registration, token generation, and access to the service provider.

At setup, all servers set up their public keys for an IND-CCA2 secure encryption scheme ENC, as well as for a signature scheme DSIG, and register them at some PKI, modeled by $\mathcal{F}_{\mathsf{CA}}$.

To register, a user generates a key-pair for a threshold encryption scheme TEnc, encrypts its password under the generated public key, and distributes the encrypted password, the (partial) secret key of the threshold encryption scheme, and some context information to each ticket-granting server. The ticket-granting servers store this information and reply with a signature send to the user, which then distributes all signatures to all ticket-granting servers, which, in turn, acknowledge account creation. Only then, the user receives an account creation confirmation. Note, however, that the ticket-granting servers may already have activated the account, even though the user which started the registration has not yet received the final acknowledgment.

A graphical overview of the registration is presented in Figure 3.10, while token generation is depicted in Figure 3.11. The presentation protocol is presented in Figure 3.12. Note, however, that the figures are dramatically simplified, as they are only meant as an overview to visualize the basic idea of the protocol.

In more detail, for token generation, the user requests the context information for the uid in question from the ticket-granting servers, including the public key of the threshold encryption scheme, and the encrypted password. The user then calculates the inverse of the password attempt pwd', and uses the homomorphic property of the threshold encryption scheme to mangle the ciphertext received from the ticket-granting servers and the ciphertext containing the password attempt. It is then randomized to "blind" the password attempt. The servers then re-randomize the blinded encrypted password quotient, and ask some external mechanism if the protocol should continue or throttle the request. If the protocols then continues, the ticket-granting servers jointly start a protocol to prove that the decryption of the blinded encrypted password quotient is equal to $1_{\mathbb{G}}$. This part of the protocol is inspired by the protocol given by Camenisch et al. [CLLN14], which, in turn, is based on Bagherzandi et al. [BJSL11]. If the password attempt was correct, the ticket-granting servers also learn this fact, and input a set of service providers S which the user is allowed to access. In the given protocol, this is realized by signatures, generated by each ticket-granting server on a bit-string. This string also contains a time-stamp which states how long the token is valid, yet also authenticates an ephemeral signature public key generated by the user.

In comparison to Camenisch et al. [CLLN14], there are some subtle differences in the password check. In particular, no secret is reconstructed. Moreover, the protocol is restricted to the case that *all* servers participate and are known beforehand, which is a reasonable assumption for SSO. Thus, the protocol is simpler. However, how to sidestep some of these restrictions is discussed in Section 3.6.

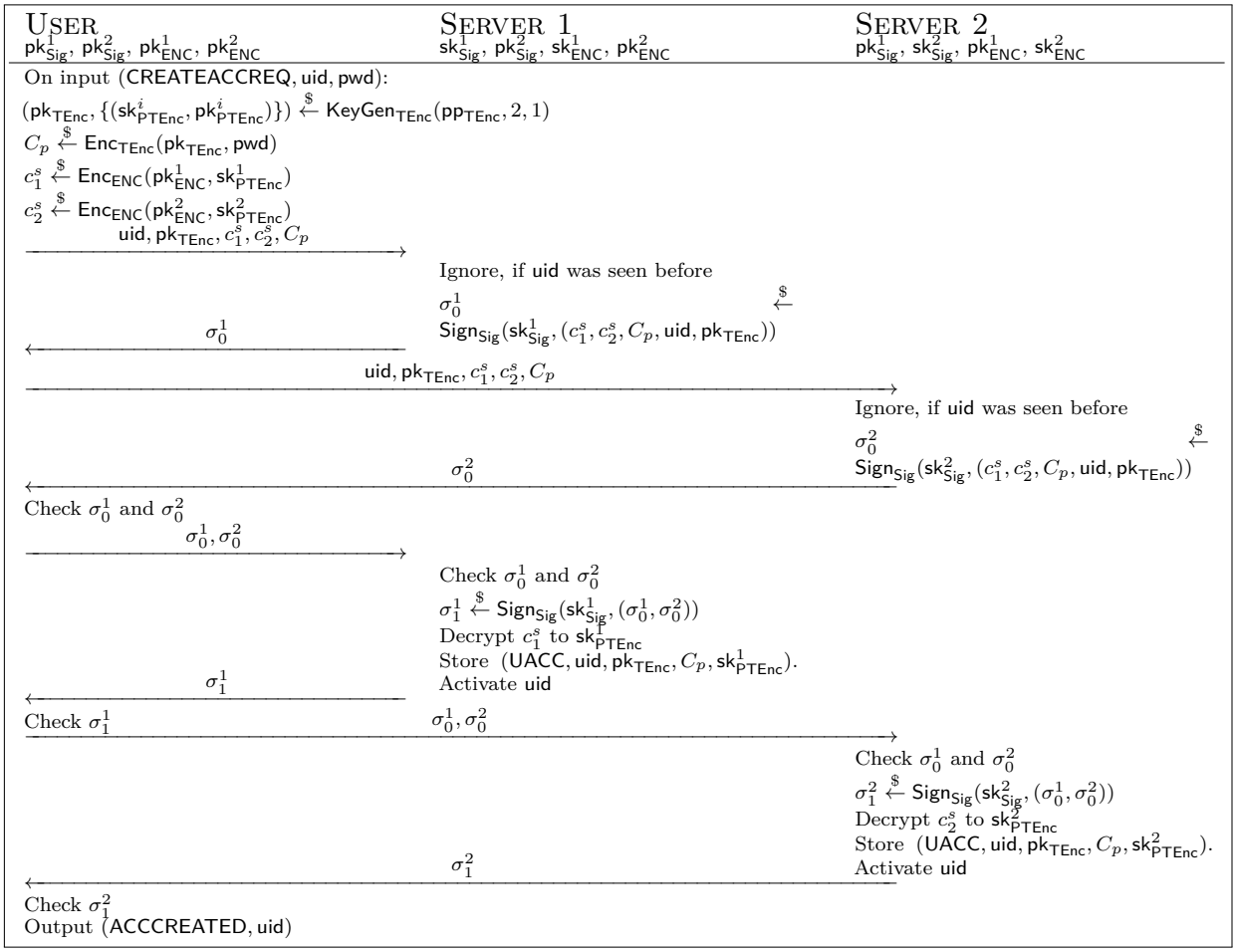

Figure 3.10: Main steps of the registration protocol for $\mathcal{F}_{\mathsf{sso}}$

For gaining access to a service provider, the user encrypts the message m to be sent to the service provider and the signatures generated by the ticket-granting servers. The user also uses the ephemeral signature private key to sign each signature and the ciphertext containing m. Furthermore, the user also states how long the generated message should be accepted by a service provider. This mechanism prohibits that an adversary "holds back" a message m for too long. The service provider simply checks the signatures and validity, decrypts the ciphertext, and then outputs the resulting message m. Thus, only if a user has received all signatures, a service will output the message, as all signatures needs to be valid. However, as the service needs to verify all signatures, it learns the messages signed.

3.2.2.1 Making the Protocol UC-Secure

To make the above sketch of the protocol UC-secure, some additional technicalities need to be deployed, e.g., there are a few more signatures generated, while each signature also needs to contain the *sid*, *qid*, and uid, to bind everything to the session and query in question. Moreover, some of the communication is encrypted to achieve the security guarantees required.

Besides from this technicalities, the main differences are two additional steps. First, all

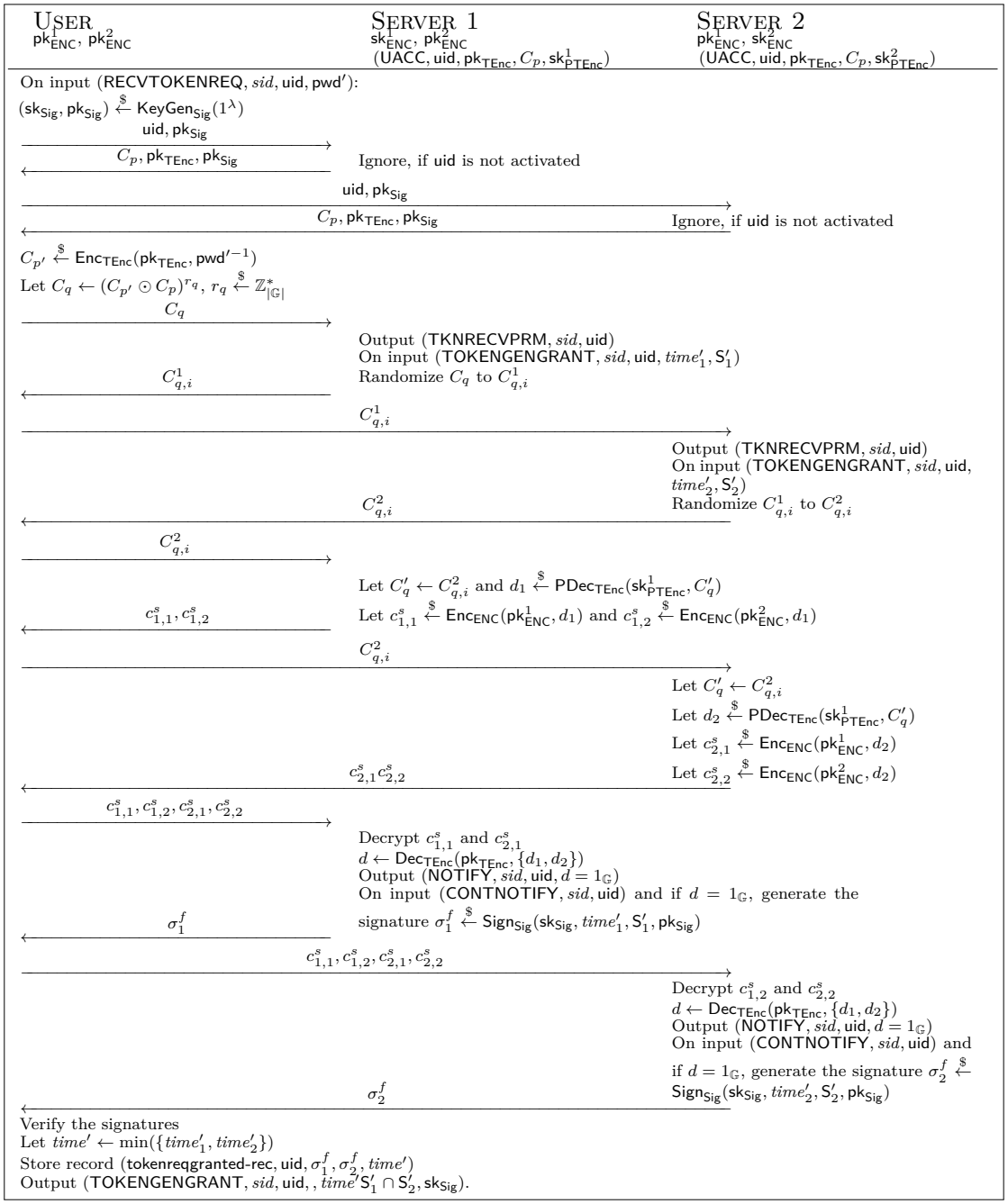

Figure 3.11: Main steps of the token generation protocol for $\mathcal{F}_{\mathsf{sso}}$

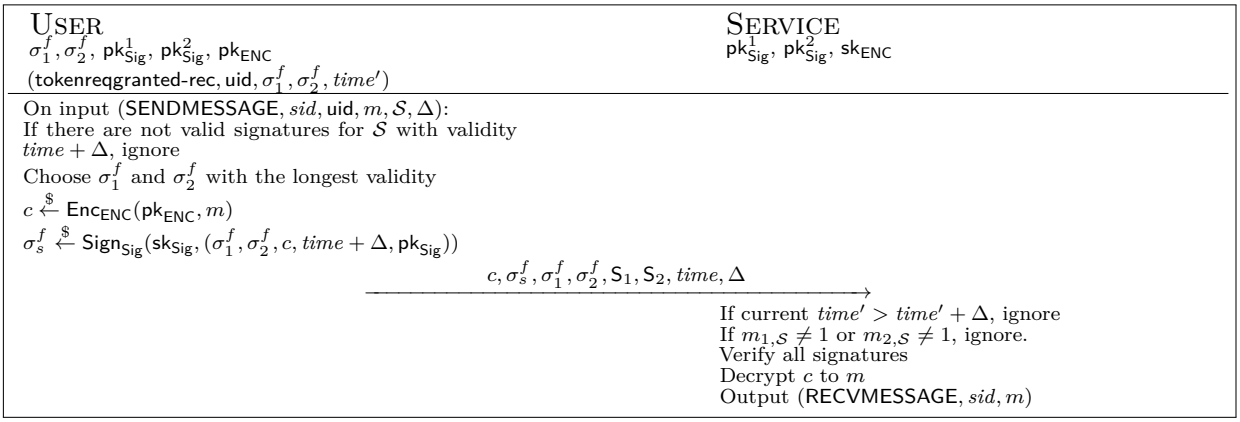

Figure 3.12: Main steps of the presentation protocol for $\mathcal{F}_{\mathsf{sso}}$

participants need to prove that they followed the steps honestly, i.e., non-interactive zero-knowledge proofs of knowledge enforce correct behavior, while they also prove knowledge of the values the simulator needs to extract, namely the password (attempts) used. This is required to reflect the adversary's behavior in the ideal world. Second, to make the simulation possible, the ticket-granting servers need to handle the decryption shares of the threshold encryption scheme TEnc in a daisy-chain before handing the decryption to $\mathcal{U}$ — in a nutshell, this step is necessary to simulate the decryption shares without knowing the actual passwords used.

Jumping ahead, in the simulation of an honest user during setup the password is not used. However, if a corrupt user later successfully performs the token generation protocol, it expects that the decryption of the randomized password quotient is $1_{\mathbb{G}}$. Thus, the simulator needs to cheat here, as it does not know the passwords involved by definition.

3.2.2.2 $\mathcal{F}_{\mathsf{CRS}}$

The ideal functionality $\mathcal{F}_{\mathsf{CRS}}$ returns the public parameters of TEnc and $\mathsf{crs}_{\mathsf{NIZKPoK}}$ of a $\mathsf{NIZKPoK}$.

In more detail, on the first call, the ideal functionality $\mathcal{F}_{\mathsf{CRS}}$ draws $\mathsf{pp}_{\mathsf{TEnc}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{TEnc}}(1^\lambda)$ and $\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda, L)$.[1] It returns $(\mathsf{pp}_{\mathsf{TEnc}}, \mathsf{crs}_{\mathsf{NIZKPoK}})$. It is stressed that the threshold encryption scheme needs to be compatible with the zero-knowledge proofs system. This can, e.g., by achieved using the instantiation given in Appendix A.

3.2.2.3 Protocol Description

In the following protocol, obvious checks, such as checking if the input (e.g., that $time \in \mathbb{N}$, the correct structuring of the sid, and the values returned by $\mathcal{F}_{\mathsf{CA}}$) is formed correctly, are omitted. This also includes that calls to $\mathcal{F}_{\mathsf{CRS}}$ and $\mathcal{F}_{\mathsf{CA}}$ are only carried out once to increase readability, while all servers ignore duplicate setup requests. Storing, however, is not possible for users, as they are assumed not to have state between protocols runs, and thus this is made explicit. Moreover, it is implicitly required that each sent message on the network is prefixed with a string corresponding to the protocol step to avoid ambiguities, e.g., (TOKEN-GEN, $2, m$),

[1] Actually, one CRS per language L is needed.

where m is the original message sent in step two of the token-generation sub-protocol. How to construct the zero-knowledge proofs used with the instantiations proposed for the building blocks is discussed in Section 3.4.

All steps in the actual protocol are pre-fixed with a number which corresponds to the ideal functionality for easier mapping.

Initialization Protocol. This step initializes the service provider S, and the ticket-granting servers T. In particular, this interface allows the environment to say that each server sets up its state, including registering public keys with $\mathcal{F}_{\mathsf{CA}}$. Note, due to UC-conventions, making this step explicit is necessary.

In more detail, each ticket-granting server is set up in the following way (which also includes authenticating public keys):

Step 3. Initialize Party ($\mathcal{T}_i$):

a) Upon input $(\mathsf{SETUPSERVER}, sid)$, generate an encryption key pair: $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.

b) Generate a key pair for a signature scheme: $(\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$. Note, the signature is not required to support efficient zero-knowledge proofs. Hence, any standard signature scheme such as RSA-FDH is sufficient.

c) Create record $(\mathsf{KEYPAIRSSOTICKET}, (\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}), (\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}))$.

d) Send $(\mathsf{REGISTER}, (\mathcal{T}, (sid, \mathcal{F}_{\mathsf{CA}})), (\mathsf{pk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{Sig}}))$ to $\mathcal{F}_{\mathsf{CA}}$, registering the public keys.

Step 4. Initialize Party ($\mathcal{S}_i$):

a) Upon input $(\mathsf{SETUPSERVER}, sid)$, generate $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.

b) Create record $(\mathsf{KEYPAIRSSOSERVICE}, (\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}))$.

c) Send $(\mathsf{REGISTER}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{CA}})), \mathsf{pk}_{\mathsf{ENC}})$ to $\mathcal{F}_{\mathsf{CA}}$.

Registration Protocol. This step allows a user $\mathcal{U}$ to create an account with the ticket-granting servers T. The user requires to enter a password pwd, which it later needs to enter again to obtain a token. For simplicity, it is assumed that the password pwd is a group element of a group defined by $\mathcal{F}_{\mathsf{CRS}}$ via $\mathsf{pp}_{\mathsf{TEnc}}$.

This can be enforced by using a collision-resistant hash-function for arbitrary bit-strings.

Step 5a. User $\mathcal{U}$ prepares messages:

a) Upon input $(\mathsf{CREATEACCREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd})$, obtain CRS $(\mathsf{pp}_{\mathsf{TEnc}}, \mathsf{crs}_{\mathsf{NIZKPoK}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.

b) Also obtain $(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{pk}^i_{\mathsf{Sig}})$ for each ticket-granting server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

c) Let $(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{i\in[1,|\mathsf{T}|]}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TEnc}}(\mathsf{pp}_{\mathsf{TEnc}}, |\mathsf{T}|, |\mathsf{T}|-1)$.

d) Encrypt the password pwd, i.e., let $(C_p; r_p) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd})$.

e) Generate a proof π_0 to prove that the content of C_p is known, bound to $n = (sid, qid, \mathsf{uid}, \mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pk}_{\mathsf{PTEnc}}, C_p)$, where $\mathsf{pk}_{\mathsf{PTEnc}} = (\mathsf{pk}^i_{\mathsf{PTEnc}})_{i\in[1,|\mathsf{T}|]}$, i.e., $\pi_0 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\mathsf{pwd}, r_p) : C_p = \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}; r_p)\}(n)$.

f) For all $i \in [1, |\mathsf{T}|]$ compute $c_i^s \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{sk}^i_{\mathsf{PTEnc}}, (n, \pi_0))$.
g) Send $(sid, qid, (c_i^s)_{i\in[1,|\mathsf{T}|]}, n, \pi_0)$ to $\mathcal{T}_1$.

Step 5b. Server $\mathcal{T}_i$ generates account:

a) Upon receiving $(sid, qid, (c_j^s)_{j\in[1,|\mathsf{T}|]}, n, \pi_0)$, ignore, if record $(\mathsf{KEYPAIRSSOTICKET}, (\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}), (\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}))$ does not exist.
b) If there is a record $(\mathsf{UACC}, sid, \mathsf{uid}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$, ignore.
c) Verify π_0. If the proof is not valid, ignore.
d) Let $\mathsf{sk}^i_{\mathsf{PTEnc}} \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c_i^s, (n, \pi_0))$. If $\mathsf{sk}^i_{\mathsf{PTEnc}} = \perp$ or $\mathsf{pk}^i_{\mathsf{PTEnc}} \neq g^{\mathsf{sk}^i_{\mathsf{PTEnc}}}$, ignore.
e) Create record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c_j^s)_{j\in[1,|\mathsf{T}|]}, C_p, \mathsf{false})$.
f) Sign $((c_j^s)_{j\in[1,|\mathsf{T}|]}, n, \pi_0, C_p)$, i.e., $\sigma_0^i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, (c_j^s)_{j\in[1,|\mathsf{T}|]}, n, \pi_0))$.
g) Send $(sid, qid, \mathsf{uid}, \sigma_0^i)$ to $\mathcal{U}$.

Step 5c. User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) Upon receiving $(sid, qid, \mathsf{uid}, \sigma_0^i)$ from $\mathcal{T}_i$, ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (c_j^s)_{j\in[1,|\mathsf{T}|]}, n, \pi_0), \sigma_0^i) = \mathsf{false}$.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (c_j^s)_{j\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 5d. User $\mathcal{U}$ receives signature from $\mathcal{T}_{|\mathsf{T}|}$:

a) After receiving the last signature, send $(sid, qid, (\sigma_i^0)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 5e. Server $\mathcal{T}_i$ receives all signatures from $\mathcal{U}$:

a) After receiving $(sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]})$, obtain $(\mathsf{pk}^j_{\mathsf{ENC}}, \mathsf{pk}^j_{\mathsf{Sig}})$ for each server $\mathcal{T}_j \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_j, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.
b) If for any $i \in [1, |\mathsf{T}|]$, $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (c_j^s)_{j\in[1,|\mathsf{T}|]}, n), \sigma_0^i) = \mathsf{false}$, ignore.
c) Generate $\sigma_1^i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}))$.
d) Update the record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c_j^s)_{j\in[1,|\mathsf{T}|]}, \mathsf{false})$ to $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c_j^s)_{j\in[1,|\mathsf{T}|]}, \mathsf{true})$.
e) Send (sid, qid, σ_1^i) to $\mathcal{U}$.

Step 5f. User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) After receiving (sid, qid, σ_1^i), ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}, n), \sigma_1^i) = \mathsf{false}$, or if such a message for qid was received before from $\mathcal{S}_i$.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_i^0)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 6. User $\mathcal{U}$ receives acknowledgment:

a) After receiving $(sid, qid, \sigma_1^{|\mathsf{T}|})$, ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}), \sigma_1^{|\mathsf{T}|}) = \mathsf{false}$, or not all all signatures have been received yet.
b) Output $(\mathsf{ACCCREATED}, sid, qid, \mathsf{uid})$, where uid, and qid, are taken from n. Note, it is required, by definition, that the user $\mathcal{U}$ has also contributed to qid, so no additional record is needed, as it is implicit.

Token Generation. This step is for the token generation. In a nutshell, a user $\mathcal{U}$ enters its uid and some password attempt pwd'. Then, if all ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ agree to proceed, i.e., do not throttle and pwd' matches the password pwd used at registration, user $\mathcal{U}$ receives a token with which is can access the service providers for a given time-frame. Which service providers $\mathcal{U}$ can access and how long a token remains valid, is input by each $\mathcal{T}_i \in \mathsf{T}$.

Step 7a. User $\mathcal{U}$ prepares messages:

a) Upon input $(\mathsf{RECVTOKENREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd})$, obtain the parameters, i.e., $(\mathsf{pp}_{\mathsf{TEnc}}, \mathsf{crs}_{\mathsf{NIZKPoK}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.

b) Also obtain $(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{pk}^i_{\mathsf{Sig}})$ for each ticket-granting server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

c) Generate a key pair for a signature scheme: $(\mathsf{sk}^{\mathcal{U}}_{\mathsf{Sig}}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.

d) Send $(sid, qid, \mathsf{uid}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}})$ to $\mathcal{T}_1$.

Step 7b. Server $\mathcal{T}_i$ sends information to $\mathcal{U}$:

a) Upon receiving $(sid, qid, \mathsf{uid}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}})$, ignore, if there is no record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c^s_j)_{j\in[1,|\mathsf{T}|]}, C_p, \mathsf{true})$.

b) Compute $\sigma^i_2 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \pi_0, C_p))$.

c) Send $(sid, qid, \sigma^i_2, n, \pi_0, C_p)$ to $\mathcal{U}$.

Step 7c. User $\mathcal{U}$ receives information from $\mathcal{T}_i$:

a) Upon receiving $(sid, qid, \sigma^i_2, n, \pi_0)$ from $\mathcal{T}_i$, ignore if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \pi_0)) = \mathsf{false}$.

b) Also ignore, if a different n, C_p or π_0 was received before.
This step essentially takes care that all ticket-granting servers have agreed on the same login information, i.e., each server sent the same set of information to the user. Thus, the user can be sure that each ticket-granting server uses the same information.

c) If $i = |\mathsf{T}|$, continue with the next step.

d) Send $(sid, qid, \mathsf{uid}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}})$ to $\mathcal{T}_{i+1}$.

Step 7d. User $\mathcal{U}$ receives information from $\mathcal{T}_{|\mathsf{T}|}$:

a) Verify π_0, and ignore if it is not valid.

b) Send $(sid, qid, (\sigma^2_i)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 8. Server $\mathcal{T}_i$ asks throttling mechanism:

a) After receiving $(sid, qid, (\sigma^2_j)_{j\in[1,|\mathsf{T}|]}, n)$, obtain $(\mathsf{pk}^j_{\mathsf{ENC}}, \mathsf{pk}^j_{\mathsf{Sig}})$ for each server $\mathcal{T}_j \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_j, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

b) If for any $j \in [1, |\mathsf{T}|]$, $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^j_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \pi_0), \sigma^2_j) = \mathsf{false}$ (n and π_0 taken from the local record), ignore.

c) Output $(\mathsf{TKNRECVPRM}, sid, qid, \mathsf{uid})$.

Step 9a. Server $\mathcal{T}_i$ proceeds:

a) On input $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$, create record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$.

b) Generate $\sigma^i_3 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, (\sigma^2_j)_{j\in[1,|\mathsf{T}|]}))$.

c) Send (sid, qid, σ_3^i) to $\mathcal{U}$.

Step 9b. User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) After receiving (sid, qid, σ_3^i), ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}^i, (sid, qid, (\sigma_j^2)_{j\in[1,|\mathsf{T}|]}, n), \sigma_3^i) = \mathsf{false}$.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_i^2)_{i\in[1,|\mathsf{T}|]}, n)$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 9c. User $\mathcal{U}$ receives signature from $\mathcal{T}_{|\mathsf{T}|}$:

a) Encrypt pwd', i.e., $(C_{p'}; r'_{\mathsf{pwd}'}) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}'^{-1})$.
b) Choose a random $r_q \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$, and calculate $C_q \leftarrow (C_{p'} \odot C_p)^{r_q}$.
c) Generate a proof π_1 that C_q has been properly re-randomized, i.e., $\pi_1 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(C_{p'}, \mathsf{pwd}', r'_{\mathsf{pwd}'}, r_q) : C_q = (C_p \odot C_{p'})^{r_q} \wedge C_{p'} \in \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}'^{-1})\}(n')$, where $n' = (n, qid, C_q, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}})$. Note, that $C_{p'}$ is not given away in clear, i.e., it remains hidden.
d) Send $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1, \perp, \perp, \perp)$ to $\mathcal{T}_1$.

Step 9d. Server $\mathcal{T}_i$ randomizes share:

a) After receiving $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1, (\sigma_i^4)_{i\in[1,i-1]}, (C_{q,i})_{i\in[1,i-1]}, (\pi_{2,i})_{i\in[1,i-1]}$ from $\mathcal{U}$, ignore, if any signature σ_i^3 or σ_i^4 is not valid.
b) Ignore, if π_1 or any $\pi_{2,i}$ is not valid.
c) Ignore, if there is no record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$.
d) Ignore, if any $C_q = (1_{\mathbb{G}}, \cdot)$. This step is necessary to avoid that the adversary uses an additively shared group exponent to make password check succeed, even though is should not.
e) If $i = 1$, let $(c_1, c_2) \leftarrow C_q$. Otherwise, set $(c_1, c_2) \leftarrow C_{q,i-1}$.
f) Choose $r_i^1 \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$ and $r_i^2 \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$. Calculate $C_{q,i} \leftarrow (g^{r_i^1} c_1^{r_i^2}, y^{r_i^1} c_2^{r_i^2})$.
g) Generate a proof $\pi_{2,i} \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r_i^1, r_i^2) : C_{q,i} = (g^{r_i^1} c_1^{r_i^2}, y^{r_i^1} c_2^{r_i^2})\}(n')$, where y is the public key from the threshold encryption scheme for user uid contained in n.
h) Sign $C_{q,i}$, i.e., let $\sigma_4^j \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (\pi_1, \pi_{2,i}, n', (\sigma_i^4)_{i\in[1,i-1]}, (C_{q,i})_{i\in[1,i]}, (\pi_{2,i})_{i\in[1,i-1]}))$.
i) Send $(sid, qid, \sigma_4^i, C_{q,i}, \pi_{2,i})$ to $\mathcal{U}$.

Step 9e. User $\mathcal{U}$ receives encryption from $\mathcal{T}_i$:

a) After receiving $(sid, qid, \sigma_4^i, C_{q,i}, \pi_{2,i})$, ignore, if the signature σ_4^i or $\pi_{2,i}$ is not valid.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1, (\sigma_i^4)_{i\in[1,i]}, (C_{q,i})_{i\in[1,i]}, (\pi_{2,i})_{i\in[1,i]})$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 9f. User $\mathcal{U}$ receives encryption from $\mathcal{T}_{|\mathsf{T}|}$:

a) Send $(sid, qid, (\sigma_4^i, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 9g. $\mathcal{T}_i$ receives all randomized shares from $\mathcal{U}$:

a) After receiving $(sid, qid, \pi_1, (\sigma_4^i, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$, verify all signatures and proofs. If not all are valid, ignore.
b) Let $C'_{q,i} \leftarrow C_{q,|\mathsf{T}|}$.
c) Ignore, if $C'_{q,i} = (1_{\mathbb{G}}, \cdot)$. This is necessary to avoid that the adversary uses a shared group exponent to make the password verification go through.

d) Compute the partial decryption share $(d_i, \pi_{d,i}) \xleftarrow{\$} \mathsf{PDec}_{\mathsf{TEnc}}(\mathsf{sk}^i_{\mathsf{PTEnc}}, C'_q, (sid, qid))$. Note, this share is bound to context (sid, qid).

e) Let $c^s_{i,j} \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}^j_{\mathsf{ENC}}, (d_i, \pi_{d,i}), (sid, qid, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \mathsf{pk}^i_{\mathsf{Sig}}))$ for all $j \in [1, |\mathsf{T}|]$.

f) Sign the shares, i.e., compute $\sigma^i_5 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, C'_q, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]}))$.

g) Send $(sid, qid, \sigma^i_5, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})$ to $\mathcal{U}$.

Step 9h. User $\mathcal{U}$ receives decryption shares from $\mathcal{T}_i$:

a) After receiving $(sid, qid, \sigma^i_5, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})$ from $\mathcal{T}_i$, ignore if σ^i_5 is invalid. C'_q is as before.

b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma^i_4, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.

c) Otherwise, continue with the next step.

Step 9i. User $\mathcal{U}$ receives decryption shares from $\mathcal{T}_{|\mathsf{T}|}$:

a) After receiving all signatures σ^i_5 and shares $c^s_{i,j}$, send $(sid, qid, (\sigma^i_5, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 11. Server $\mathcal{T}_k$ checks password attempt:

a) After receiving $(sid, qid, (\sigma^i_5, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$, check that all signatures are valid. If not, ignore.

b) Decrypt each $c^s_{k,j}$, i.e., $(d_j, \pi_{d,j}) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c^s_{k,j}, (sid, qid, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \mathsf{pk}^j_{\mathsf{Sig}}))$. If decryption outputs $\perp$, ignore.

c) If $\mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}^i_{\mathsf{PTEnc}}, d_j, \pi_{d,j}, C'_q, (sid, qid)) = \mathsf{false}$ for any $(d_j, \pi_{d,j})$, ignore.

d) If $\mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_j)_{1\leq j\leq|\mathsf{T}|}, C'_q)$ is equal to $1_{\mathbb{G}}$, output $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathsf{true})$. Output $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathsf{false})$ otherwise.

Step 12a. Server $\mathcal{T}_i$ continues:

a) On input $(\mathsf{CONTNOTIFY}, sid, qid, \mathsf{uid})$, let the message $m_{\mathcal{T}_i} \leftarrow (sid, qid, \mathsf{uid}, b_{\mathcal{T}_i,1}, b_{\mathcal{T}_i,2}, \ldots, b_{\mathcal{T}_i,|\mathsf{S}|}, time'_{\mathcal{T}_i}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}})$, where $b_{\mathcal{T}_i,j} \leftarrow 1$, if $\mathcal{S}_j \in \mathsf{S}'$, and $b_{\mathcal{T}_i,j} \leftarrow 0$ otherwise. The values are stored in the record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time'_i, \mathsf{S}'_i)$.

b) If $1_{\mathbb{G}} \neq \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_i)_{i\in[1,n]}, C'_q)$, ignore.

c) Sign $m_{\mathcal{T}_i}$, i.e., $\sigma^f_i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m_{\mathcal{T}_i})$.

d) Generate $\sigma^i_6 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, \sigma^f_i, m_{\mathcal{T}_i}, (d_i, \pi_{d,i})_{i\in[1,|\mathsf{T}|]}, time'_i, \mathsf{S}'_i))$.

e) Send $(sid, qid, \sigma^f_i, m_i, (d_i, \pi_{d,i})_{i\in[1,|\mathsf{T}|]}, time'_i, \mathsf{S}'_i, \sigma^i_6)$ to $\mathcal{U}$.

Step 12b. User $\mathcal{U}$ receives token from $\mathcal{T}_i$:

a) After receiving $(sid, qid, \sigma^f_i, m_i, (d_i, \pi_{d,i})_{i\in[1,|\mathsf{T}|]}, time'_i, \mathsf{S}'_i, \sigma^i_6)$, check that the signatures σ^f_i and σ^i_6 are valid, and ignore if this is not the case.

b) If a different $(d_j, \pi_{d,j})_{j\in[1,|\mathsf{T}|]}$ was received before, ignore.

c) If $\mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}^j_{\mathsf{PTEnc}}, d_j, \pi_{d,j}, C'_q, (sid, qid)) = \mathsf{false}$ for any $(d_j, \pi_{d,j})$, ignore.

d) If $i \neq |\mathsf{T}|$, send $(sid, (\sigma^i_5, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.

e) Otherwise, continue with the next step.

Step 13. User $\mathcal{U}$ receives token from $\mathcal{T}_{|\mathsf{T}|}$:

a) If any $\pi_{d,i}$ is invalid, ignore.

b) If $1_{\mathbb{G}} \neq \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_i)_{1\leq i\leq|\mathsf{T}|}, C'_q)$, ignore.

c) After receiving the last token, create record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, (m_{\mathcal{T}_i}, \sigma_i^f)_{i\in[1,|\mathsf{T}|]}, time', \mathsf{S}', \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}})$, where $time' = \min(\bigcup_{i\in[1,|\mathsf{T}|]} time_i')$, and $\mathsf{S}' = \{\mathcal{S}_j \mid \forall m_{\mathcal{T}_i} : b_{\mathcal{T}_i,j} = 1\}$.
d) Output $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$.

Gain Access to Service Provider. This step is to actually gain access to a service provider.

Step 14. User $\mathcal{U}$ sends message to $\mathcal{S}_k$:
a) On input $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, m, \mathcal{S}_k, \Delta)$, get the current time *time*.
b) If there exists a record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid', \mathcal{U}, \mathsf{uid}, (m_{\mathcal{T}_i}, \sigma_i^f)_{i\in[1,|\mathsf{T}|]}, (time_i')_{i\in[1,|\mathsf{T}|]}, time', \mathsf{S}', \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}})$ with $\mathcal{S}_k \in \mathsf{S}'$, and $time' > time + \Delta$, proceed. Otherwise, ignore.
c) Obtain $\mathsf{pk}_{\mathsf{ENC}}^{\mathcal{S}_k}$ for $\mathcal{S}_k$ by sending $(\mathsf{RETRIEVE}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.
d) Encrypt m, i.e., let $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}^{\mathcal{S}_k}, m, (sid, qid, \mathsf{uid}, time + \Delta))$
e) Generate $\sigma_s^f \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}^{\mathcal{U}}, (sid, qid, \mathsf{uid}, (\sigma_i^f, m_{\mathcal{T}_i})_{i\in[1,|\mathsf{T}|]}, \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}}, c, \mathcal{S}, time, \Delta))$.
f) Send $(sid, qid, \mathsf{uid}, \sigma_s^f, (\sigma_i^f, m_{\mathcal{T}_i})_{i\in[1,|\mathsf{T}|]}, c, time, \Delta, \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}})$ to $\mathcal{S}$.

Step 15. Service $\mathcal{S}_k$ received message from $\mathcal{U}$:
a) On input $(sid, qid, \mathsf{uid}, \sigma_s^f, (\sigma_i^f, m_{\mathcal{T}_i})_{i\in[1,|\mathsf{T}|]}, c, time', \Delta', \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}})$, obtain $(\mathsf{pp}_{\mathsf{TEnc}}, \mathsf{crs}_{\mathsf{NIZKPoK}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.
b) If there is no record $(\mathsf{KEYPAIRSSOSERVICE}, (\mathsf{sk}_{\mathsf{ENC}}^{\mathcal{S}_k}, \mathsf{pk}_{\mathsf{ENC}}^{\mathcal{S}_k}))$, ignore.
c) Obtain $(\mathsf{pk}_{\mathsf{ENC}}^{i}, \mathsf{pk}_{\mathsf{Sig}}^{i})$ for each server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.
d) Get the current time *time*. If $time' + \Delta' < time$, ignore.
e) Verify each σ_i^f w.r.t. $\mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}}$ and $\mathsf{pk}_{\mathsf{Sig}}^{i}$. If not all are valid, ignore.
f) Verify σ_s^f w.r.t. $\mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}}$. If it is not valid, ignore.
g) If not for all $m_{\mathcal{T}_i}$, $b_{\mathcal{T}_i,j} = 1$, where j is the index of $\mathcal{S} \in \mathsf{S}$, ignore.
h) If not for all $m_{\mathcal{T}_i}$, $time_i \geq time$, ignore.
i) If not all all *qid*s in the $m_{\mathcal{T}_i}$ are the same, ignore.
j) Decrypt c, i.e., let $m \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}^{\mathcal{S}_k}, c, (sid, qid, \mathsf{uid}, time'))$. If decryption fails, ignore.
k) Output $(\mathsf{RECVMESSAGE}, sid, qid, \mathsf{uid}, m)$.

The proof of the following Theorem is given in Section 3.2.3.

Theorem 3.10. *The protocol described above securely implements the ideal functionality $\mathcal{F}_{\mathsf{sso}}^1$ in the $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid model, if* DSIG, ENC, TEnc *and* NIZKPoK *are secure, assuming static corruptions.*

The ticket-granting servers may decide to allow for access to different service providers and a different token validity. This is reflected in the protocol by using a set of signatures presented to the service providers. However, one may ask the question if there is way to reduce this overhead. A solution is simple: if all ticket-granting servers share a key of a threshold signature scheme [DF89], agree on the same *time*, and set of service providers $\mathsf{S}' \subseteq \mathsf{S}$ the user $\mathcal{U}$ can access, one signature is enough. The sharing of the secret key can be done at the initialization of the ticket-granting servers using, e.g., UC-secure multi-party computation [Lin09]. Even though this may seem expensive at first glance, this initial step only has to be performed once,

and thus amortizes over time. Another possibility are aggregate signatures [BGLS03], where all signatures generated are aggregated into a single shorter one. However, with a realistic amount of ticket-granting servers, say two or three, the presented approach is still efficient enough from a practical perspective, and keeps the description of the protocol short. Thus, it depends on the concrete scenario whether one wants to deploy the altered schemes — altering the protocol and ideal functionality is straightforward.

Likewise, some additional improvements can be integrated. For example, an already seen zero-knowledge proof does not need to be verified again. Moreover, in the defined protocol the user $\mathcal{U}$ servers as the central hub, distributing the messages sent across the network. Clearly, the servers itself can form a "daisy-chain", skipping the user to decrease network delays. However, in this case an honest server needs to verify signatures and proofs, which is, in the above protocol, is "out-sourced" to the user. Thus, it depends on the concrete scenario which version is preferable.

needs to verify signatures and proofs, which is, in the above protocol, is out-source

How to further change the protocol to account for additional use-cases is discussed in Section 3.6.

3.2.3 Security of Protocol 1

High-Level Idea. The proof of Theorem 3.10 is done by providing a simulator and a sequence of games. The initial game is the real experiment and the final game (Game 12) runs the simulator given only the information that is also available to the adversary when interacting with the ideal functionality.

In a nutshell, the proof idea is as follows. In the first game, the real-world is executed, where a challenger still receives all inputs and outputs from the honest parties. With each game-hop, the protocol is gradually made less depended on the input received for the honest party and prove why this does not change the view of the adversary in a non-negligible way. Eventually, in the final hop, the protocol can be simulated with the given simulator, barely based on the information it receives from the ideal functionality which, in turn, proves security.

Next, a rough sketch of the sequence of games is given. The detailed description, as well as the description of the simulator, is given afterwards.

Overview over Hops. The proof consist of a sequence of games that a challenger runs with the real-world adversary. In the final game the transition, the challenger internally runs the ideal functionality $\mathcal{F}_{\mathsf{sso}}$, and simulates all messages based merely on the information it can obtain from $\mathcal{F}_{\mathsf{sso}}$. First, the hops are sketched to ease understanding of the full proof.

Game 1 allows the challenger to abort, if the adversary was able to forge a signature. Game 2 and Game 3 allows the adversary to simulate proofs. Game 4 allows the challenger to extract from proofs and forces the adversary to generate the proofs correctly. Game 5 to Game 7 change the way how encryptions of the password are done, while Game 8 and Game 9 exchange the encryptions sent by honest parties to honest parties by dummy values, while Game 10 shows that even with a simulated setup the adversary receives valid decryption shares. Finally, Game 11 proves that an adversary cannot generate tokens without the help of the ticket-granting servers. In Game 12, the transition to the ideal world is done, where a simulator can run only interacting with the ideal functionality and the real adversary.

Detailed Sequence of Games. Now, each Game i is described in more detail, while it is also argued why the view of the environment does not significantly change.

Game 0: In the first game, the challenger executes the real protocol for *all* honest players, obtaining their inputs from, and passing the respective outputs to, the environment. It is already assumed that the challenger keeps all values generated for the honest parties, and input from (and output to) each honest participant. Clearly, this is only an internal change and does not change the view of the adversary at all.

Game 1: Here, the challenger aborts, if there it sees a signature valid under a $\mathsf{pk}^i_{\mathsf{Sig}}$ of an honest ticket-granting server for a message m^* which has never been signed. In particular, these are the signatures σ^i_0, σ^i_1, σ^i_2, σ^i_3, σ^i_4, σ^i_5 and σ^i_6, for each honest ticket-granting server $\mathcal{T}_i \in \mathsf{T}$.

Clearly, the hop behaves differently to the prior game, if the abort happens. However, an adversary distinguishing between the two games was able to generate a forged signature σ^*, which breaks the unforgeability of the underlying signature scheme. Thus, by a standard hybrid argument, this hop is indistinguishable. In particular, the reduction receives $\mathsf{pk}_{\mathsf{Sig}}$, and sequentially embeds it as one of the (honest) server's public keys $\mathsf{pk}^i_{\mathsf{Sig}}$. The signatures to be generated are provided by the signing oracle. This embedding does not change the view of the adversary so far. Then, if the adversary was able to generate a forged signature, the reduction can simply return it.

Game 2: The challenger now replaces $\mathsf{crs}_{\mathsf{NIZKPoK}}$ with the one generated by the ZK-simulator. An adversary distinguishing this hop can trivially be turned into an adversary against the proof system. The reduction is simple: if the adversary can distinguish this hop, the reduction receives $\mathsf{crs}_{\mathsf{NIZKPoK}}$ from its own challenger and embeds it into the values returned by $\mathcal{F}_{\mathsf{CRS}}$. Whatever the adversary outputs, is also output by the reduction.

Game 3: The challenger now uses the ZK-simulator to fake, i.e., simulate, all proofs done by the honest parties. Clearly, a distinguisher can be turned against an adversary against the proof system. In particular, the reduction receives the challenge $\mathsf{crs}_{\mathsf{NIZKPoK}}$, embeds into the values returned by $\mathcal{F}_{\mathsf{CRS}}$ and uses the proof oracle to generate all proofs. The value output by the environment is also output by the reduction. Note, $\mathsf{crs}_{\mathsf{NIZKPoK}}$ is distributed as in the prior hops by definition, and thus no additional argument is needed.

Game 4: The challenger now extracts the values knowledge is proven about from the proofs given by the adversary. If this is not possible, the challenger aborts. All extracted values are kept internally. So far, this hop is indistinguishable by the soundness of the proof system. The reduction is simple: the challenger embeds $\mathsf{crs}_{\mathsf{NIZKPoK}}$ provided by its own challenger as $\mathsf{crs}_{\mathsf{NIZKPoK}}$. That is, if the adversary was able to generate a proof which cannot be extracted, it simply outputs this proofs as its own forgery. The challenger also aborts, if there are any inconsistencies; the challenger can simply recalculate the steps done in the protocol, and check if the adversary performed the steps honestly. If, however, there are inconsistencies, the proof can simply be returned to its own challenger; no further steps are necessary.

This hop is valid, if the $\mathsf{NIZKPoK}$ is used as a black-box with the definitions. If, however, the proof-system is instantiated with the more efficient Fiat-Shamir-transform, with the alterations presented in Section 3.4, one has to resort to a different argument. Namely, assume that the adversary $\mathcal{A}$ was able to fake a proof and thus the protocol behaves differently to what is expected, e.g., passwords verify while they should not or the challenger cannot extract the values encrypted to the sky. In this case, the reduction can simply embed $\mathsf{crs}_{\mathsf{NIZKPoK}}$ received from its own challenger, and return a randomly chosen proof π^* (along with the statement proven, which can be derived from the protocol). Note, this random choice is necessary, as there protocol steps where the reduction cannot decide whether the proof was generated correctly, e.g., the proofs $\pi_{2,i}$ if the passwords in question, as this implies a DDH-solver. See Step 8 and Step 9. Due to this random choice, however, the reduction succeeds only with probability $1/q_\pi$, where q_π is the number of proofs the adversary sends. However, this is still non-negligible. This especially means, that the values required to be extracted by the simulator are actually extractable, as required for the simulator given in next section.

Game 5: The next change is how the retrieval is performed by an honest user. In particular, the challenger knows whether a retrieval attempt is done with the correct pwd', as pwd can be extracted from C_p in the case of a corrupt user during setup, and is known for an honest user anyway. In particular, if the passwords match, i.e., if $\mathsf{pwd} = \mathsf{pwd}'$, C_q is a new encryption of $1_{\mathbb{G}}$, as well as $C_{q,i}$. As the proofs are already simulated, this does not change the view of the adversary at all, as the distributions are equal due to the randomness added. However, if the password does not match, C_q and $C_{q,i}$ are replaced with fresh encryptions of random values not equal to $1_{\mathbb{G}}$. As the distributions are perfectly equal, the adversary does not see any difference.

Game 6: Now is changed how the retrieval is performed by a corrupt user if not all ticket-granting servers are corrupt. In particular, the challenger knows whether a retrieval attempt is done with the correct pwd', as pwd can be extracted from C_p in the case of a corrupt user during setup, and is known for an honest user, and pwd' can be extracted from π_1. In particular, if the passwords match, i.e., if $\mathsf{pwd} = \mathsf{pwd}'$, then replace $C_{q,i}$ with a fresh encryption of $1_{\mathbb{G}}$.

So far, the distributions are equal. If, however, $\mathsf{pwd} \neq \mathsf{pwd}'$, then replace $C_{q,i}$ with an encryption of a random element (but $1_{\mathbb{G}}$). Again, as the distributions are perfectly equal, while the proofs are already simulated, the adversary cannot see any difference.

Game 7: The challenger aborts, if the "last" ticket-granting server, i.e., $\mathcal{T}_{|\mathsf{T}|}$ is honest, but the protocol does not output a token, but $\mathsf{pwd} = \mathsf{pwd}'$ was the case. As this means that the group exponent was hit, as the only case that the protocol aborts prematurely and artificially, even if the adversary played honestly, while the last ticket-granting server re-randomizes completely. This only happens by chance, i.e., $1/|\mathbb{G}|$.

Note, the case that the adversary makes the protocol can be ignored, as the adversary can make the protocol fail anyway.

Game 8: In this hop, the challenger replaces the c_i^s send from an honest user to an honest server by encryptions of 1 (for each server). As usual, a standard hybrid argument shows, that

this hop is (so far) indistinguishable due to the IND-CCA2 security of the encryption scheme. However, there are two additional cases which have to be considered. First, the adversary may replace (some) c_i^s with an invalid partial decryption key. In this case, the ticket-granting server aborts (as in the real protocol). In the case that c_i^s decrypts to a correct partial secret key, one can use this adversary to break the semantic security of the TEnc.

In particular, the reduction knows t secret shares, and the adversary (with a non-negligible chance), provides an additional key. Then, the reduction can simply decrypt c_i^s to gain an additional (with non-negligible probability) partial secret key, and decrypting the challenge c^* from the semantic security challenger. Note, however, that the embedding of the challenge public keys $\mathsf{pk}_{\mathsf{TEnc}}$, and $\mathsf{pk}^i_{\mathsf{PTEnc}}$ is random, and can only be used in one setup session. This, however, still only implies a polynomial loss overall. In particular, only one challenge is given for multiple registration attempts, and thus all but one are honestly generated.

Note, any ciphertext c_i^s provided by the adversary can simply be decrypted using the decryption oracle provided. Moreover, the corresponding partial public keys are bound to the context n, and can thus not be re-used in a different contexts. Now, the only option left is that n was changed as well. In this case, the ticket-granting servers simply follow the protocol. Note, the case where an account was already created, the servers ignore the message anyway, and can thus be ignored.

Game 9: In this hop, the challenger replaces the $c^s_{i,j}$ from an honest ticket-granting server encrypted for a honest ticket-granting server by encryptions of 1. As usual, a standard hybrid argument shows, that this hop is (so far) indistinguishable due to the IND-CCA2 security of the encryption scheme.

Note, the adversary cannot replace ciphertexts, as the shares are signed, and signature forgeries are already excluded. Thus, they cannot be re-used in different contexts, due to the attached label, as the *qid*s are unique. Moreover, also due to the attached label, any ciphertexts provided by the adversary can simply be decrypted using the decryption oracle provided.

Game 10: The challenger now changes the way how the threshold encryption (and decryption) work, if not all ticket-granting servers are corrupt. In particular, in the case of an honest user during setup, the challenger replaces C_p with an encryption of $1_{\mathbb{G}}$. Clearly, as the adversary knows (at most) $|\mathsf{T}| - 1$ partial secret keys, an adversary distinguishing the replacement trivially can be turned against an adversary against the semantic security of the TEnc used using a series of hybrids, i.e., by embedding the challenge keys into one of the setup sessions (and sending the known partial secret keys to the corrupt ticket-granting servers) one by one.

Now, if a honest user tries to get a token for an account created by an honest user, the challenger proceeds as before.

In the case of a corrupt user which tries to get a token for an account created by a honest user, however, the challenger needs to make the decryption shares look correct, if not all ticket-granting servers are corrupt. The challenger knows whether $\mathsf{pwd} = \mathsf{pwd}'$ (pwd' can be extracted from π_1), and proceeds as follows.

Each honest ticket-granting server no longer randomizes the share $C_{q,i}$ honestly, but returns are fresh encryption of $1_{\mathbb{G}}$ is the passwords match and of a random group element otherwise. As the distributions are now correct again, the adversary cannot notice any difference.

Game 11: The challenger now aborts, if a service outputs a message from a corrupt user, even though the user in question never received a (valid) token from the ticket-granting servers, if at least one ticket-granting server is honest. As this implies that there is at least one σ_j^f (or σ_s^f) for some message for which the ticket-granting servers or user have never issued a signature, this tuple breaks the unforgeability of the signature scheme (note, honest users do not receive any signatures, as the encryptions are already simulated, and the *qid*s are unique).

A reduction in the first case is simple: $\mathsf{pk}_{\mathsf{Sig}}$ is embedded as one of the ticket-granting servers' one. Then, each signature is generated using the the signature-generation oracle provided. This does not change the view of the adversary so far. Thus, unforgeability of the signature scheme is broken, as σ_i^f and the corresponding message are always given to the service provider in any case. In the second case, i.e., where the adversary was able to generate a signature σ_s^f in the name of a user as the only possibility left, the reduction is the same. The challenge public key is embedded as one of the users in the system. Then, whenever the adversary generates a new σ_s^f under the challenge public key, this forgery can simply be returned.

Note, the messages can be extracted in all case, as the decryption keys are known.

Game 12: Now, the challenger switches to the ideal world, and simply simulates the protocol with the information provided by the ideal functionality $\mathcal{F}_{\mathsf{sso}}$ alone. The simulator is provided in the following. This is only an internal change, and does not change the view of the adversary.

3.2.3.1 Simulator

Now, the simulator SIM is presented. The simulator is split up into four parts to ease readability. Note, the phrase "the protocol is followed honestly" means according to the presented game-hops.

3.2.3.2 Initialization Protocol

The initialization simulation simply follows the protocol, including $\mathcal{F}_{\mathsf{CA}}$ in all cases.

3.2.3.3 Registration Protocol

The simulation of the registration part is split into different cases, depending which parties are corrupt, and if there already is an ongoing registration.

Honest User $\mathcal{U}$. Firstly, the case of an honest user is described.

There is at least one honest ticket-granting server $\mathcal{T}_i \in \mathsf{T}$. Here, the simulator SIM receives the input $(\mathsf{CREATEACCREQ}, sid, qid, \mathcal{U}, \mathsf{uid})$. It follows the protocol honestly, with the exception that is simulates the proofs and encrypts 0s as the ciphertexts c_i^s send to the simulated honest parties (Game 11), while the threshold encryption contains $1_{\mathbb{G}}$ (Game 13). Note, the honest servers no longer decrypt the ciphertexts. If, however, the adversary chose to alter sent values received to one of the honest servers, there are two cases. Firstly, the values received by the honest server(s) are changed completely, i.e., the adversary has overruled the setup request for *qid*. In this case, the simulator can extract pwd' from π_0, and can send $(\mathsf{CREATEACCREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd}')$ to $\mathcal{F}_{\mathsf{sso}}^1$, and can then continue

simulating according to the protocol (See below for the case of a corrupt user). Note, simply exchanging ciphertexts is not possible, due to Game 10. The second case, i.e., the case where the values are not consistent (n or c_i^s have changed for only some of the honest servers) the account on the servers is "blocked", as there is now no way to successfully register an account, i.e., the simulated servers will never continue the request (not even a new one due to the same uid; same *qid*s are dropped anyway), while simulated users never continue due to the changed values, as the servers never generate the required signatures (See Game 2).

When the last honest server $\mathcal{T}_i$ sends (sid, σ_1^i, n), the account is created, which is acknowledged towards $\mathcal{F}_{\mathsf{sso}}^1$ by sending $(\mathsf{ACCCREATED}, sid, qid, \mathcal{U}, \mathsf{uid})$. Note, this means that the account is created on the servers, and can be used, while the user with uid did not receive any output yet.

Finally, after the honest user receives all correct values, the simulator allows $\mathcal{U}$ to output $(\mathsf{ACCCREATED}, sid, qid, \mathsf{uid})$ in the ideal world.

All ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ are corrupt. In this case, the simulator SIM receives the input $(\mathsf{CREATEACCREQ}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathsf{pwd})$, and follows the normal protocol. Whenever the simulated user $\mathcal{U}$ would output $(\mathsf{ACCCREATED}, sid, qid, \mathsf{uid})$, the simulator SIM triggers the output in the ideal world as well.

Corrupt User $\mathcal{U}$. Now, the case of a corrupt user is described.

There is at least one honest ticket-granting server $\mathcal{T}_i$. In this case, a honest server receives $(sid, (c_j^s)_{j\in[1,|\mathsf{T}|]}, n, \pi_0)$, and follows the protocol, but sends $(\mathsf{CREATEACCREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd})$ to $\mathcal{F}_{\mathsf{sso}}$ on the first reception of the sent message at the first honest ticket-granting server. Note, pwd can be extracted from π_0. The simulator directly receives back control, and follows the protocol once more. If the "last" simulated server $\mathcal{T}_i$ receives $(sid, (\sigma_i^0)_{i\in[1,|\mathsf{T}|]}, n)$, and is willing to continue (i.e., the protocol was followed correctly and there is no other registration with the same uid ongoing), then the simulator sends $(\mathsf{ACCCREATED}, sid, qid, \mathcal{A}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{sso}}$, and sends the last message to the corrupt user.

All ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ are corrupt. This case is internal to the adversary $\mathcal{A}$ and thus does not need to be simulated.

3.2.3.4 Simulation of the Token-Generation Protocol

Next, the simulation of the token-generation protocol is presented. As before, it is split into several parts, depending on which parties are corrupt, and which party initiated the protocol.

Honest User $\mathcal{U}$. Firstly, the case of an honest user is described.

There is at least one honest ticket-granting server $\mathcal{T}_i$. Here, the simulator SIM is activated upon receiving $(\mathsf{RECVTOKENREQ}, sid, qid, \mathcal{U}, \mathsf{uid})$. SIM starts the protocol, simulating the (honest) ticket-granting servers and the user $\mathcal{U}$ as described in the protocol. The simulation is done honestly up to the point where the first honest server $\mathcal{T}_i$ outputs

(TKNRECVPRM, *sid*, *qid*, uid). This can happen if either all messages sent where received honestly, or the adversary interfered. The case of non-interference is described first. In the case the adversary overruled the messages, SIM continues as in the case with a corrupt user $\mathcal{U}$, described below.

The simulation proceeds as in the protocol, till all honest servers agreed to participate (i.e., if SIM received (TOKENGENGRANT, *sid*, *qid*, uid, $\mathcal{T}$) for all ticket-granting servers). Then, SIM sends (TESTPW, *sid*, *qid*, $\perp$) to $\mathcal{F}_{\mathsf{sso}}$, receiving (TESTPW, *sid*, *qid*, b), where $b = \mathsf{true}$ or $b = \mathsf{false}$. It then continues simulation according to the protocol till the user must generate C_q — the simulator encrypts $1_{\mathbb{G}}$ if $b = \mathsf{true}$ and a random value if $b = \mathsf{false}$ (Game 8). It then proceeds as in the protocol, but replaces $C_{q,i}$ with encryptions of random group elements if $b = \mathsf{false}$ (Game 8). If $b = \mathsf{true}$, it replaces each $C_{q,i}$ with encryptions of $1_{\mathbb{G}}$ (Game 8), and replaces the encryptions $c^s_{i,j}$ with 0 (Game 12). It then follows the protocol honestly, including the case when a simulated ticket-granting server $\mathcal{T}_i$ outputs (NOTIFY, *sid*, *qid*, uid, b). The simulator SIM then mimics this behavior in the ideal world by sending (NOTIFY, *sid*, *qid*, $\mathcal{U}$, uid, $\mathcal{T}_i$) to $\mathcal{F}_{\mathsf{sso}}$, and then follows the protocol honestly again. Finally, when the simulated user $\mathcal{U}$ outputs (TKNRECVCOMPL, *sid*, *qid*, uid, *time*′, S′), SIM also mimics the behavior in the ideal world by sending (TKNRECVCOMPL, *sid*, *qid*, $\mathcal{U}$, uid, *time*′, S′) to $\mathcal{F}_{\mathsf{sso}}$. Note, *time*′ and S′ may depend on the input from the corrupt servers, but can be derived in the same manner as in the protocol.

All ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ are corrupt. Here, the simulator SIM is activated on input (RECVTOKENREQ, *sid*, *qid*, $\mathcal{U}$, uid). The simulator first simply follows the protocol, till it receives n from all servers. It then obtains pwd′, extractable due to π_0. It then sends (TESTPW, *sid*, *qid*, pwd′) to $\mathcal{F}_{\mathsf{sso}}$ to directly retrieve (TESTPW, *sid*, *qid*, b). If $b = \mathsf{false}$, it follows the protocol with a different pwd (i.e., a random ciphertext), and in the case $b = \mathsf{true}$ with the decrypted password. If, at some later point in time, the simulated user would output (TKNRECVCOMPL, *sid*, *qid*, uid, *time*′, S′), the simulator mimics the same output in the ideal world by sending (TKNRECVCOMPL, *sid*, *qid*, $\mathcal{U}$, uid, *time*′, S′) to $\mathcal{F}_{\mathsf{sso}}$, where *time*′ and S′ are derived as in the protocol.

Corrupt User $\mathcal{U}$. Now, the case of a corrupt user is described.

There is at least one honest ticket-granting server $\mathcal{T}_i$. Here, the simulator follows the protocol honestly, till the (first) simulated $\mathcal{T}_i$ outputs (TKNRECVPRM, *sid*, *qid*, uid). It then follows the protocol honestly (including the case where the simulated ticket-granting servers agree to participate, as described above), till the first simulated ticket-granting server $\mathcal{T}_j$ receives C_q. It then extracts pwd from π_1, and mimics the behavior in the ideal world by sending (RECVTOKENREQ, *sid*, *qid*, uid, pwd) to $\mathcal{F}_{\mathsf{sso}}$, directly receiving (RECVTOKENREQ, *sid*, *qid*, $\mathcal{A}$, uid).

The simulation now branches, depending on whether the account uid was created by corrupt user or was a simulated setup.

Account was created by a honest user. The simulator follows the protocol honestly, till it needs to provide the opening to the commitments. Namely, SIM sends (TESTPW, *sid*, *qid*, $\perp$) to $\mathcal{F}_{\mathsf{sso}}$, receiving (TESTPW, *sid*, *qid*, b), where $b = \mathsf{true}$ or $b = \mathsf{false}$.

Depending on b, the simulator generates the openings and partial decryption shares according to Game 13, i.e., making the decryption shares look correct in the case $b = \mathsf{true}$, even though C_p contains $1_{\mathbb{G}}$, and random elements in the case $b = \mathsf{false}$. Before the last ticket-granting server sends the last value to the network, the simulator then grants the token in the ideal world by sending $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathcal{U}, \mathsf{uid}, time', \mathsf{S}')$ to $\mathcal{F}_{\mathsf{sso}}$, directly receiving back control, where $time'$ and S' are generated as in the protocol.

Account was created by a corrupted user. In the case, the simulator already knows pwd used at setup, and can thus follow the protocol honestly, and simply mimics the behavior in the ideal world, i.e., once the last honest ticket-granting server sends out its message, SIM also sends $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathsf{uid}, \infty, \mathsf{S})$ to $\mathcal{F}_{\mathsf{sso}}$.

All ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ are corrupt. This step is completely internal to the adversary, and thus does not need to be simulated.

3.2.3.5 Simulation of the Messaging Protocol

Honest User $\mathcal{U}$. Firstly, the case of an honest user is described.

$\mathcal{S}$ *is honest.* Here, the simulator SIM receives the input $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, \mathcal{U}, \mathcal{S}, \mathsf{S}', time'', \mathsf{S}'', \Delta)$. It then simply follows the protocol with the values stored from the token-generation simulation. Note, this also means that the "most recent" token is used. If then the simulated service receives the message sent not altered before anything else with that qid, SIM sends $(\mathsf{RECVMESSAGE}, sid, qid)$ to $\mathcal{F}_{\mathsf{sso}}$, which triggers the output of the message in the ideal world. If, however, the adversary sends a message $(sid, c', qid, \mathsf{uid}, time', \Delta')$ which would make the simulated service output a message m', SIM decrypts c' (to obtain m'), and sends $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}', m', \mathcal{S}, \Delta')$ to $\mathcal{F}_{\mathsf{sso}}$, which directly returns $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, \mathcal{U}, \mathcal{S}, \mathsf{S}', time'', \mathsf{S}'', \Delta)$. The simulator SIM then sends $(\mathsf{RECVMESSAGE}, sid, qid)$ to $\mathcal{F}_{\mathsf{sso}}$ to trigger the output of the new message in the ideal world. This case can happen, if the adversary obtained a token for some uid (which is still valid), and overrules the sent message. This is not avoidable, as the other tokens remain valid — note, an honest user will never "overrule" such a sent message, as it uses a new qid. Note, in the case all ticket-granting servers are corrupt, the adversary can sent any message for uid - in the case not all ticket-granting servers are corrupt, the adversary needs to have a token for that uid; the simulated service $\mathcal{S}$ would abort otherwise.

$\mathcal{S}$ *is corrupt.* Here, the simulator receives $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, \mathcal{S}, \mathsf{S}', time'', \mathsf{S}'', m, \Delta)$, and simply follows the protocol honestly.

Corrupt User $\mathcal{U}$. Now, the case of a corrupt user is described.

$\mathcal{S}$ *is honest.* Here, the adversary sends a message $(sid, c', qid, \mathsf{uid}, time', \Delta')$ which makes the simulated service output a message m', i.e., all checks pass. SIM decrypts c' (to obtain m'), and sends $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}', m', \mathcal{S}, \Delta')$ to $\mathcal{F}_{\mathsf{sso}}$, which directly

5. `Create Account Request.` On input (CREATEACCREQ, *sid*, *qid*, uid, pwd) from party $\mathcal{U}$:
   - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$ or $time = -1$, or a record (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, $\cdot$, $\cdot$) exists, ignore.
   - Create record (user-rec, *sid*, *qid*, $\mathcal{U}$, uid, pwd, false).
   - If $\mathcal{U} = \mathcal{A}$ or all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, create record (uidcorr-rec, *sid*, uid).
   - If all $\mathcal{T}_i \in \mathsf{T}$ are corrupt, send (CREATEACCREQ, *sid*, *qid*, $\mathcal{U}$, uid, pwd) to $\mathcal{A}$.
   - Send (CREATEACCREQ, *sid*, *qid*, $\mathcal{U}$, uid) to $\mathcal{A}$.

Figure 3.13: Changes to the registration interfaces for $\mathcal{F}_{\mathsf{sso}}^2$

returns (SENDMESSAGE, *sid*, *qid*, uid$'$, $\mathcal{S}'$, S$'$, *time*$''$, S$'$, m', Δ'). The simulator SIM then sends (RECVMESSAGE, *sid*, *qid*) to $\mathcal{F}_{\mathsf{sso}}$ to trigger the output in the ideal world.

Note, in the case all ticket-granting servers are corrupt, the adversary can sent any message for uid - in the case not all ticket-granting servers are corrupt, the adversary needs to have a token for that uid; the simulated service $\mathcal{S}$ would abort otherwise.

$\mathcal{S}$ is corrupt. This case is completely internal to the adversary $\mathcal{A}$, and thus no simulation is necessary.

This completes the description of the simulator SIM.

3.3 The Privacy-Enhanced Protocol

In this section, the functionality $\mathcal{F}_{\mathsf{sso}}^2$ is presented. In a nutshell, $\mathcal{F}_{\mathsf{sso}}^2$ behaves as $\mathcal{F}_{\mathsf{sso}}^1$, but offers better privacy guarantees, but with slightly less efficient realization. As the functionalities are rather similar, only their differences are discussed.

Namely, $\mathcal{F}_{\mathsf{sso}}^2$ hides which other service providers a user can access, how long a token is valid from a service provider, while the user can also resort to establish different identities with each service provider and also hides from which token-generation run the token used in derived from. This allows one to use the protocol in environments which require additional privacy guarantees.

3.3.1 The Differences to $\mathcal{F}_{\mathsf{sso}}^1$

Only the second, third and fourth group of the interfaces change, i.e., the setup interfaces serve the same purpose in both protocols.

The Registration Interfaces. The registration interfaces, depicted in Figure 3.13, changes only slightly, i.e., the first interface in that group.

5. The interface CREATEACCREQ now also keeps track whether an account was ever under adversarial control. This is necessary to correctly model that an adversary can then link pseudonyms, as it learns the corresponding secret key in the real world. Thus, it must be reflected in the ideal world as well.

12. `Continue Notify.` On input $(\mathsf{CONTNOTIFY}, sid, qid, \mathsf{uid})$ from party $\mathcal{T}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{T} \notin \mathsf{T}$, or $time = -1$, ignore.
    - If there is no record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, \mathsf{true})$, ignore.
    - If there is a record $(\mathsf{srvnot\text{-}rec}, sid, qid)$, ignore.
    - Create record $(\mathsf{srvnot\text{-}rec}, sid, qid)$.
    - If $\mathcal{U}$ is corrupt, send $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathcal{T}, time', \mathsf{S})$ to $\mathcal{A}$, where $time'$, and S, are taken from record $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}, time', \mathcal{S}')$.
    - Send $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathcal{T})$ to $\mathcal{A}$.
13. `Token Generation Complete.` On input $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathcal{U}, \mathsf{uid}, time'', \mathsf{S}'')$ from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
    - If all $\mathcal{T}_i \in \mathsf{T}$ are honest, let $\mathsf{S}'' \leftarrow \mathsf{S}$ and $time'' \leftarrow \infty$.
    - If there is a record of the form $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$, ignore.
    - If not for all honest $\mathcal{T}_i \in \mathsf{T}$ there exists a record $(\mathsf{srvcont\text{-}rec}, sid, qid, \mathsf{uid}, \mathcal{T}_i, \mathsf{true})$, ignore.
    - For all existing records of the form $(\mathsf{srvprocreq\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, \mathcal{T}_i, time_i, \mathsf{S}_i)$, let $time' \leftarrow \min(\{time_i\} \cup time'')$, and $\mathsf{S}' \leftarrow \bigcap \mathsf{S}_i \cap \mathsf{S}''$.
    - Create record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, time', \mathsf{S}', (time_i, \mathsf{S}_i)_{i\in[1,n]})$.
    - If $\mathcal{U} = \mathcal{A}$, create record $(\mathsf{uidcorr\text{-}rec}, sid, \mathsf{uid})$.
    - Output $(\mathsf{TKNRECVCOMPL}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$ to $\mathcal{U}$.

Figure 3.14: Changes to the token generation interfaces for $\mathcal{F}_{\mathsf{sso}}^2$

The Token Generation Interfaces. The token generation interfaces, depicted in Figure 3.14, changes only slightly, i.e., only the last interface in that group.

12. The interface TKNRECVCOMPL now also keeps track whether an adversary was able to successfully guess a password.
13. The interface CONTNOTIFY now only leaks the access rights and the information how long the token generated is valid, if the user initiating the token generation is corrupt.

Now is explained how the communication interfaces change.

14. The SENDMESSAGE interface now also allows a user $\mathcal{U}$ to enter a scope string scope. Moreover, the adversary no longer receives the uid, the time difference Δ or the set of service providers the user can access.
15. The RECVMESSAGE interface now also handles pseudonyms. The pseudonyms N_i are drawn randomly by the functionality, based on the uid, the service provider $\mathcal{S}$ to be accessed and the scope string scope. Note, however, that the resulting pseudonym is then fixed once and for all (w.r.t. the service provider/uid/scope combination), as long as one ticket-granting server $\mathcal{T}_i \in \mathsf{T}$ remains honest.
    Namely, if all ticket-granting servers are corrupt, the adversary can decide that new pseudonyms are drawn or are re-used from another user. This is intrinsic, as the adversary may hand a bogus pseudonym secret-key to a user at token generation. Moreover, the drawn pseudonyms and the scope strings are hidden from the adversary, as long as the service provider $\mathcal{S}$ to be accessed is not corrupt, i.e., those values are hidden from the adversary.
    Note, however, that the pseudonyms are not scope-exclusive. The reason for this is benign; scope-exclusive pseudonyms are deterministic and thus there cannot exist a simulator which

14. `Send Message.` On input (SENDMESSAGE, sid, qid, uid, m, $\mathcal{S}$, Δ, scope) from party $\mathcal{U}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{S} \notin \mathsf{S}$, $time = -1$, or $\Delta \notin \mathbb{N}$, ignore.
    - If there is a record (msg-rec, sid, qid, $\mathcal{U}$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$), where $\mathcal{U} \neq \mathcal{A}$, ignore.
    - If there is no record (tokenreqgranted-rec, sid, qid', $\mathcal{U}$, uid, $time'$, S'), where $\mathcal{S} \in \mathsf{S}'$, and $time + \Delta \leq time'$, ignore, if not all $\mathcal{T}_i \in \mathsf{T}$ and $\mathcal{U}$ are corrupt.
    - If all $\mathcal{T}_i \in \mathsf{T}$ and $\mathcal{U}$ are corrupt, set $time'' \leftarrow \infty$, and $\mathsf{S}'' \leftarrow \mathsf{S}$.
    - Create record (msg-rec, sid, qid, $\mathcal{U}$, uid, m, $\mathcal{S}$, $time + \Delta$, scope, false).
    - If there is a record (uidcorr-rec, sid, uid) and $\mathcal{S}$ is corrupt, send (SENDMESSAGE, sid, qid, uid, $\mathcal{U}$, $\mathcal{S}$, m, Δ, scope) to $\mathcal{A}$.
    - If $\mathcal{S}$ is corrupt, send (SENDMESSAGE, sid, qid, ℓ(uid, $\mathcal{S}$, scope), $\mathcal{U}$, $\mathcal{S}$, m, Δ, scope) to $\mathcal{A}$.
    - Send (SENDMESSAGE, sid, qid, $\mathcal{U}$, $\mathcal{S}$) to $\mathcal{A}$.
15. `Receive Message.` On input (RECVMESSAGE, sid, qid, $\mathsf{PRENYM}_{\mathcal{A}}$) from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, or $time = -1$, ignore.
    - If there is a record (msg-rec, sid, qid, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, true), ignore.
    - If there is no record (msg-rec, sid, qid, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, $\cdot$, false), ignore.
    - Update the record (msg-rec, sid, qid, $\mathcal{U}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, false) to (msg-rec, sid, qid, $\mathcal{A}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, true), where $\mathcal{U} \neq \mathcal{A}$, if it exists.
    - If $time > time'$, ignore, if there does not exist a record (msg-rec, sid, qid, $\mathcal{A}$, uid, m, $\mathcal{S}$, n, S'', $\cdot$, $\cdot$, false), where $n \geq time$.
    - Update record (msg-rec, sid, qid, $\mathcal{A}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, false) to (msg-rec, sid, qid, $\mathcal{A}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, true), if it exists.
    - If $\mathsf{PRENYM}_{\mathcal{A}} \neq \perp$, ignore if there is an honest $\mathcal{T}_i \in \mathsf{T}$, and $\mathcal{U} \neq \mathcal{A}$.
    - If $\mathsf{PRENYM}_{\mathcal{A}} = \perp$, and a record (scope-rec, sid, uid, $\mathcal{S}$, scope, NYM', $\perp$) exists, let $\mathsf{NYM} \leftarrow \mathsf{NYM}'$. Otherwise, create that record with a random (not yet chosen) $\mathsf{NYM}' \xleftarrow{\$} \mathcal{N}$.
    - If $\mathsf{PRENYM}_{\mathcal{A}} \neq \perp$, and a record (scope-rec, sid, uid, $\mathcal{S}$, scope, NYM', $\mathsf{PRENYM}_{\mathcal{A}}$) exists, let $\mathsf{NYM} \leftarrow \mathsf{NYM}'$. Otherwise, create that record with a random (not yet chosen) $\mathsf{NYM}' \xleftarrow{\$} \mathcal{N}$.
    - Ignore, if not all ticket-granting servers $\mathcal{T}_i \in \mathsf{T}$ are corrupt, but two records (scope-rec, sid, uid, $\cdot$, $\cdot$, $\cdot$, $\mathsf{PRENYM}_{\mathcal{A}}$) with different uids exist.
    - If a record (msg-rec, sid, qid, $\mathcal{A}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, true) exists, take scope, m and uid from this record. Otherwise take it from record (msg-rec, sid, qid, $\mathcal{U}$, uid, m, $\mathcal{S}$, $time''$, S'', $time'$, scope, true).
    - Output (RECVMESSAGE, sid, qid, NYM, m, scope) to $\mathcal{S}$.

Figure 3.15: Communication interfaces for $\mathcal{F}_{\mathsf{sso}}^2$. $\mathcal{N}$ denotes some pseudonym space. The leakage function ℓ, on input uid, $\mathcal{S}$ and scope, behaves as follows. It returns a list of all qids from the msg-rec records, where there are records of the form (msg-rec, sid, qid, $\cdot$, uid, $\cdot$, $\mathcal{S}$, $\cdot$, scope, $\cdot$).

produces a "succinct" usk for the adversary which "magically" makes all generated pseudonyms match.

The Corruption Interfaces. Due to the incorporated pseudonym system, the simulator needs some additional information once the adversary was able to successfully receive a token for a user uid it did not create. Thus, there is an additional interface designed to capture this behavior, depicted in Fig. 3.16.

16. The CORRSCOPES interface allows the adversary to obtain the information necessary to

16. `Corrupt`. On input (CORRSCOPES, sid, uid) from adversary $\mathcal{A}$:
    - If $sid \neq (\mathsf{T}, \mathsf{S}, sid')$, $\mathcal{S} \notin \mathsf{S}$, $time = -1$, or $\Delta \notin \mathbb{N}$, ignore.
    - If there is no record (uidcorr-rec, sid, uid), ignore.
    - Send a list L to $\mathcal{A}$, where L is of the form (scope-rec, sid, qid, scope), taken from all records of the form (msg-rec, sid, qid, $\cdot$, uid, $\cdot$, $\cdot$, $\mathcal{S}$, $\cdot$, scope, $\cdot$), where $\mathcal{S}$ is corrupt.

Figure 3.16: Corruption interfaces $\mathcal{F}_{\mathsf{sso}}^2$.

simulate the pseudonyms correctly. Namely, as soon as the adversary gains knowledge of usk, it can recalculate pseudonyms and can thus "break" unlinkability, as it can now simply re-calculate the used pseudonyms with the now learned secret key.

3.3.2 Realizing Protocol for $\mathcal{F}_{\mathsf{sso}}^2$

Now, the second protocol realizing $\mathcal{F}_{\mathsf{sso}}^2$ is presented. Due to the better privacy guarantees, the protocol is slightly less efficient than the one for $\mathcal{F}_{\mathsf{sso}}^1$, as it involves more zero-knowledge proofs.

3.3.2.1 Protocol Description

The protocol is derived from the first protocol. However, due to the additional possibilities, there are some differences which are explained next. In more detail, these are the changes to $\mathcal{F}_{\mathsf{CRS}}$, and the protocol itself.

Changes to $\mathcal{F}_{\mathsf{CRS}}$. The common reference string functionality $\mathcal{F}_{\mathsf{CRS}}$ now, in addition to $\mathsf{crs}_{\mathsf{NIZKPoK}}$ and $\mathsf{pp}_{\mathsf{TEnc}}$ returns the public parameters $\mathsf{pp}_{\mathsf{NYM}}$, where $\mathsf{pp}_{\mathsf{NYM}} = (g_c, h_c)$, where $g_c \xleftarrow{\$} \mathbb{G}^\times$ and $h_c \xleftarrow{\$} \mathbb{G}^\times$ (like Pedersen), and the public parameters $\mathsf{pp}_{\mathsf{SS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SS}}(1^\lambda, |\mathsf{T}|)$ for the secret sharing building block, i.e., the CRS now consists of $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \mathsf{pp}_{\mathsf{NYM}}, \mathsf{pp}_{\mathsf{SS}}, \mathsf{pp}_{\mathsf{TEnc}})$. Moreover, it is required that uid is some compatible group element.

If the signature scheme by Groth is used (See Appendix B), $\mathcal{F}_{\mathsf{CRS}}$ also needs to return the corresponding public parameters.

Changes to the Initialization Protocol. The initialization routines remain mostly unchanged, with the exception that the service providers create an additional key κ for a PRF, which is used to make pseudonyms simulatable.

In more detail, a service provider is set up in the following way:

Step 3. Initialize Party ($\mathcal{S}_i$):

a) Upon input (SETUPSERVER, sid), generate: $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.
b) Generate a key for a PRF: $\kappa \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{PRF}}(1^\lambda)$.
c) Create record (KEYPAIRSSOSERVICE, $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}), \kappa$).
d) Send (REGISTER, $(\mathcal{S}, (sid, \mathcal{F}_{\mathsf{CA}})), \mathsf{pk}_{\mathsf{ENC}}$) to $\mathcal{F}_{\mathsf{CA}}$.

Changes to the Registration Protocol. The changes to the registration protocol are also subtle. Namely, a user additionally generates a key pair $(\mathsf{usk}, \mathsf{upk})$ for a pseudonym-system, and secret shares the secret key usk among the ticket-granting servers. The secret sharing is necessary to maintain the unlinkability of the pseudonyms. At retrieval, the secret key usk is reconstructed and is used to generate the pseudonyms for the service providers. However, notice that the pseudonym public key upk cannot be re-used for a different account $\mathsf{uid}' \neq \mathsf{uid}$, as the ticket-granting servers forbid multiple accounts with same already seen upk.

Step 5a. Registration – User $\mathcal{U}$ prepares messages:

a) Upon input $(\mathsf{CREATEACCREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd})$, obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \mathsf{pp}_{\mathsf{NYM}}, \mathsf{pp}_{\mathsf{SS}}, \mathsf{pp}_{\mathsf{TEnc}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.

b) Also obtain $(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{pk}^i_{\mathsf{Sig}})$ for each server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

c) Let $(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{i\in[1,|\mathsf{T}|]}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TEnc}}(\mathsf{pp}_{\mathsf{TEnc}}, |\mathsf{T}|, |\mathsf{T}|-1)$.

d) Let $\mathsf{usk} \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathsf{upk} = g_c^{\mathsf{usk}}$.

e) Encrypt the password pwd, i.e., let $(C_p; r_p) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd})$.

f) Let $\{s_1, s_2, \ldots, s_n\} \xleftarrow{\$} \mathsf{Share}_{\mathsf{SS}}(m)$.

g) Generate a proof π_0 to prove that the content of C_p is known, bound to context $n = (sid, qid, \mathsf{uid}, \mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pk}_{\mathsf{PTEnc}}, C_p, \mathsf{upk})$, where $\mathsf{pk}_{\mathsf{PTEnc}} = (\mathsf{pk}^i_{\mathsf{PTEnc}})_{i\in[1,|\mathsf{T}|]}$, i.e., let the proof π_0 generated as follows: $\pi_0 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\mathsf{pwd}, r_p) : C_p = \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}; r_p)\}(n)$.

h) For all $i \in [1, |\mathsf{T}|]$ compute $c^s_i \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}^i_{\mathsf{ENC}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, s_i), (n, \pi_0))$.

i) Send $(sid, qid, (c^s_i)_{i\in[1,|\mathsf{T}|]}, n, \pi_0)$ to $\mathcal{T}_1$.

Step 5b. Registration – Server $\mathcal{T}_i$ generates account:

a) Upon receiving $(sid, qid, (c^s_j)_{j\in[1,|\mathsf{T}|]}, n, \pi_0)$, ignore, if record $(\mathsf{KEYPAIRSSOTICKET}, (\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}), (\mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}))$ does not exist.

b) If there is a record $(\mathsf{UACC}, sid, \mathsf{uid}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$, ignore.

c) If there is a record $(\mathsf{UACC}, sid, \cdot, \cdot, n, \cdot, \cdot, \cdot, \cdot, \cdot)$, where n contains upk, ignore.

d) Verify π_0. If the proof is not valid, ignore.

e) If $\mathsf{pk}^i_{\mathsf{PTEnc}} \neq g^{\mathsf{sk}^i_{\mathsf{PTEnc}}}$, ignore.

f) Let $\mathsf{sk}^i_{\mathsf{PTEnc}} \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c^s_i, (n, \pi_0))$. If $\mathsf{sk}^i_{\mathsf{PTEnc}} = \perp$, ignore.

g) Create record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, s_i, (c^s_j)_{j\in[1,|\mathsf{T}|]}, C_p, \mathsf{false})$.

h) Sign sid, and $((c^s_j)_{j\in[1,|\mathsf{T}|]}, n, \pi_0)$, i.e., $\sigma^i_0 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, (c^s_j)_{j\in[1,|\mathsf{T}|]}, n, \pi_0))$.

i) Send $(sid, qid, \mathsf{uid}, \sigma^i_0)$ to $\mathcal{U}$.

Step 5c. Registration – User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) Upon receiving $(sid, qid, \mathsf{uid}, \sigma^i_0)$, ignore, if σ^i_0 is not valid.

b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (c^s_j)_{j\in[1,|\mathsf{T}|]}, n, \pi_0)$ to $\mathcal{T}_{i+1}$.

c) Otherwise, continue with the next step.

Step 5d. Registration – User $\mathcal{U}$ receives signature from $\mathcal{T}_{|\mathsf{T}|}$:

a) After receiving the last signature, send all signatures to $\mathcal{T}_1$, i.e., send $(sid, qid, (\sigma^0_i)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 5e. Registration – Server $\mathcal{T}_i$ receives all signatures from $\mathcal{U}$:

a) After receiving $(sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}, n)$, obtain $(\mathsf{pk}^j_{\mathsf{ENC}}, \mathsf{pk}^j_{\mathsf{Sig}})$ for each server $\mathcal{T}_j \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_j, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

b) If for any $i \in [1, |\mathsf{T}|]$, $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, ((c_j^s)_{j\in[1,|\mathsf{T}|]}, n), \sigma_0^i) = \mathsf{false}$, ignore.

c) Generate $\sigma_1^i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}))$.

d) Update record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, s_i, (c_j^s)_{j\in[1,|\mathsf{T}|]}, C_p, \mathsf{false})$ to $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c_j^s)_{j\in[1,|\mathsf{T}|]}, C_p, \mathsf{true})$.

e) Send (sid, qid, σ_1^i) to $\mathcal{U}$.

Step 5f. Registration – User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) After receiving the message(sid, qid, σ_1^i) from $\mathcal{S}_i$, ignore, if the signature does not verify, i.e., if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, ((\sigma_j^0)_{j\in[1,|\mathsf{T}|]}), \sigma_1^i) = \mathsf{false}$.

b) Also ignore, if such a message for qid was received before from $\mathcal{S}_i$.

c) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_i^0)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.

d) Otherwise, continue with the next step.

Step 6. Registration – User $\mathcal{U}$ receives acknowledgment:

a) After receiving $(sid, qid, \sigma_1^{|\mathsf{T}|})$ from $\mathcal{T}_{|\mathsf{T}|}$, ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (\sigma_j^0)_{j\in[1,|\mathsf{T}|]}), \sigma_1^{|\mathsf{T}|}) = \mathsf{false}$.

b) Ignore, if not all signatures have been received yet.

c) Output $(\mathsf{ACCCREATED}, sid, qid, \mathsf{uid})$, where uid, and qid, are taken from n. Note, it is required that $\mathcal{U}$ also contributed to qid, so no additional record is needed, as it is implicit.

Changes to the Token Generation Protocol. This step shows how the token generation protocol is altered. As for the first protocol, it checks the password attempt pwd', and also allows the ticket-granting servers to throttle ongoing token generation attempts.

However, this step now also reconstructs the secret-key usk for a pseudonym-system NYM, which can, at token presentation, generate a pseudonym nym for a string scope. This sub-protocol is very similar to the one presented for $\mathcal{F}^1_{\mathsf{sso}}$.

Moreover, a user no longer needs to generate a signature key pair, but an ephemeral encryption key used to privately receive the shares of usk and the signatures $\sigma^f_{\mathcal{T}_i,\mathcal{S}_j}$ from each ticket-granting server. Thus, the ephemeral $\mathsf{pk}_{\mathsf{Sig}}$ is no longer signed and thus is no longer signed, i.e., σ_i^f no longer protects this value.

Note, the signature scheme used for generating $\sigma^f_{\mathcal{T}_i,\mathcal{S}_j}$ now needs to support efficient zero-knowledge proofs, i.e., all other signatures can still be standard ones, i.e., one can choose the most efficient version available.

Step 7a. User $\mathcal{U}$ prepares messages:

a) Upon input $(\mathsf{RECVTOKENREQ}, sid, qid, \mathsf{uid}, \mathsf{pwd})$, ask $\mathcal{F}_{\mathsf{CRS}}$ to obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \mathsf{pp}_{\mathsf{NYM}}, \mathsf{pp}_{\mathsf{SS}}, \mathsf{pp}_{\mathsf{TEnc}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.

b) Also obtain $(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{pk}^i_{\mathsf{Sig}})$ for each server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

c) Generate a key pair for a labeled CCA2-secure encryption scheme ENC: $(\mathsf{sk}^{\mathcal{U}}_{\mathsf{ENC}}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.

d) Send $(sid, qid, \mathsf{uid}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{ENC}})$ to $\mathcal{T}_1$.

Step 7b. Server $\mathcal{T}_i$ sends information $\mathcal{U}$:

a) Upon receiving $(sid, qid, \mathsf{uid}, \mathsf{pk}_{\mathsf{ENC}})$, ignore, if there is no record $(\mathsf{UACC}, sid, \mathsf{uid}, n, \pi_0, \mathsf{sk}^i_{\mathsf{PTEnc}}, (c^s_j)_{j\in[1,|\mathsf{T}|]}, s_i, C_p, \mathsf{true})$.

b) Compute $\sigma^i_2 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}^{\mathcal{U}}_{\mathsf{ENC}}, \pi_0, C_p))$.

c) Send $(sid, qid, \sigma^i_2, n, \pi_0, C_p)$ to $\mathcal{U}$.

Step 7c. User $\mathcal{U}$ receives information from $\mathcal{T}_i$:

a) Upon receiving $(sid, qid, \sigma^i_2, n, \pi_0)$, ignore if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}_{\mathsf{ENC}}, \pi_0), \sigma^i_2) = \mathsf{false}$.

b) Also ignore, if a different n, C_p or π_0 was received before.

c) If $i = |\mathsf{T}|$, continue with the next step.

d) Send $(sid, qid, \mathsf{uid}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{ENC}})$ to $\mathcal{T}_{i+1}$.

Step 7d. User $\mathcal{U}$ receives information from $\mathcal{T}_{|\mathsf{T}|}$:

a) Verify π_0, and ignore if it is not valid.

b) Send $(sid, qid, (\sigma^2_i)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 8. Server $\mathcal{T}_i$ asks throttling mechanism:

a) After receiving $(sid, qid, (\sigma^2_j)_{j\in[1,|\mathsf{T}|]})$, obtain $(\mathsf{pk}^j_{\mathsf{ENC}}, \mathsf{pk}^j_{\mathsf{Sig}})$ for each server $\mathcal{T}_j \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_j, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.

b) If for any $j \in [1, n]$, $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^j_{\mathsf{Sig}}, (sid, qid, \mathsf{uid}, n, \mathsf{pk}^{\mathcal{U}}_{\mathsf{ENC}}, \pi_0), \sigma^2_j) = \mathsf{false}$ (n and π_0 taken from the local record), ignore.

c) Output $(\mathsf{TKNRECVPRM}, sid, qid, \mathsf{uid})$.

Step 9a. Server $\mathcal{T}_i$ proceeds:

a) On input $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$, create record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$.

b) Generate $\sigma^i_3 \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, (\sigma^2_j)_{j\in[1,|\mathsf{T}|]}))$.

c) Send (sid, qid, σ^i_3) to $\mathcal{U}$.

Step 9b. User $\mathcal{U}$ receives signature from $\mathcal{T}_i$:

a) After receiving (sid, qid, σ^i_3) from $\mathcal{T}_i$, ignore, if $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^i_{\mathsf{Sig}}, (sid, qid, (\sigma^2_j)_{j\in[1,|\mathsf{T}|]}), \sigma^i_3) = \mathsf{false}$.

b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma^2_i)_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.

c) Otherwise, continue with the next step.

Step 9c. User $\mathcal{U}$ receives signature from $\mathcal{T}_{|\mathsf{T}|}$:

a) Encrypt pwd', i.e., $(C_{p'}; r'_{\mathsf{pwd}'}) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}'^{-1})$.

b) Choose a random $r_q \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$, and calculate $C_q \leftarrow (C_{p'} \odot C_p)^{r_q}$.

c) Generate a proof π_1 that C_q has been properly re-randomized, i.e., $\pi_1 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(C_{p'}, \mathsf{pwd}', r'_{\mathsf{pwd}'}, r_q) : C_q = (C_p \odot C_{p'})^{r_q} \wedge C_{p'} \in \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}'^{-1})\}(n')$, where $n' = (n, qid, C_q, \mathsf{pk}_{\mathsf{ENC}})$.

d) Send $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1)$ to $\mathcal{T}_1$.

Step 9d. Server $\mathcal{T}_i$ randomizes share:

a) After receiving $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1, (\sigma_i^4)_{i\in[1,i-1]}, (C_{q,i})_{i\in[1,i-1]}, (\pi_{2,i})_{i\in[1,i-1]}$ from $\mathcal{U}$, ignore, if any signature σ_i^3 or σ_i^4 is not valid.
b) Ignore, if π_1 or any $\pi_{2,i}$ is not valid.
c) Ignore, if there is no record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time', \mathsf{S}')$.
d) Ignore, if any $C_q = (1_{\mathbb{G}}, \cdot)$. This step is necessary to avoid that the adversary uses an additively shared group exponent to make password check succeed, even though is should not.
e) If $i = 1$, let $(c_1, c_2) \leftarrow C_q$. Otherwise, set $(c_1, c_2) \leftarrow C_{q,i-1}$.
f) Choose $r_i^1 \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$ and $r_i^2 \xleftarrow{\$} \mathbb{Z}^*_{|\mathbb{G}|}$. Calculate $C_{q,i} \leftarrow (g^{r_i^1} c_1^{r_i^2}, y^{r_i^1} c_2^{r_i^2})$.
g) Generate a proof $\pi_{2,i} \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r_i^1, r_i^2) : C_{q,i} = (g^{r_i^1} c_1^{r_i^2}, y^{r_i^1} c_2^{r_i^2})\}(n')$, where y is the public key from the threshold encryption scheme for user uid contained in n.
h) Sign $C_{q,i}$ and the received signatures, i.e., let $\sigma_4^j \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (\pi_1, \pi_{2,i}, n', (\sigma_i^4)_{i\in[1,i-1]}, (C_{q,i})_{i\in[1,i]}, (\pi_{2,i})_{i\in[1,i-1]}))$.
i) Send $(sid, qid, \sigma_4^i, C_{q,i}, \pi_{2,i})$ to $\mathcal{U}$.

Step 9e. User $\mathcal{U}$ receives encryption from $\mathcal{T}_i$:

a) After receiving $(sid, qid, \sigma_4^i, C_{q,i}, \pi_{2,i})$, ignore, if the signature σ_4^i or $\pi_{2,i}$ is not valid.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_i^3)_{i\in[1,|\mathsf{T}|]}, C_q, n', \pi_1, (\sigma_i^4)_{i\in[1,i]}, (C_{q,i})_{i\in[1,i]}, (\pi_{2,i})_{i\in[1,i]})$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 9f. User $\mathcal{U}$ receives encryption from $\mathcal{T}_{|\mathsf{T}|}$:

a) Send $(sid, qid, (\sigma_4^i, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 9g. $\mathcal{T}_i$ receives all randomized shares from $\mathcal{U}$:

a) After receiving $(sid, qid, \pi_1, (\sigma_4^i, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$, verify all signatures and proofs. If not all are valid, ignore.
b) Let $C'_{q,i} \leftarrow C_{q,|\mathsf{T}|}$.
c) Ignore, if $C'_{q,i} = (1_{\mathbb{G}}, \cdot)$. This is necessary to avoid that the adversary uses a shared group exponent to make the password verification go through.
d) Compute the partial decryption share $(d_i, \pi_{d,i}) \xleftarrow{\$} \mathsf{PDec}_{\mathsf{TEnc}}(\mathsf{sk}^i_{\mathsf{PTEnc}}, C'_q, (sid, qid))$. Note, this share is bound to context (sid, qid).
e) Let $c^s_{i,j} \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}^j_{\mathsf{ENC}}, (d_i, \pi_{d,i}), (sid, qid, \mathsf{pk}^{\mathcal{U}}_{\mathsf{Sig}}, \mathsf{pk}^i_{\mathsf{Sig}}))$ for all $j \in [1, |\mathsf{T}|]$.
f) Sign the shares, i.e., compute $\sigma_5^i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, C'_q, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]}))$.
g) Send $(sid, qid, \sigma_5^i, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})$ to $\mathcal{U}$.

Step 9h. User $\mathcal{U}$ receives decryption shares from $\mathcal{T}_i$:

a) After receiving $(sid, qid, \sigma_5^i, (c^s_{i,j})_{j\in[1,|\mathsf{T}|]})$ from $\mathcal{T}_i$, ignore if σ_5^i is invalid. C'_q can be calculated from the shares known.
b) If $i \neq |\mathsf{T}|$, send $(sid, qid, (\sigma_4^i, C_{q,i}, \pi_{2,i})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.
c) Otherwise, continue with the next step.

Step 9i. User $\mathcal{U}$ receives decryption shares from $\mathcal{T}_{|\mathsf{T}|}$:

a) After receiving all signatures σ_5^i and shares $c_{i,j}^s$, send $(sid, qid, (\sigma_5^i, (c_{i,j}^s)_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_1$.

Step 11. Server $\mathcal{T}_k$ checks password attempt:

a) After receiving $(sid, qid, (\sigma_5^i, (c_{i,j}^s)_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$, check that all signatures are valid. If not, ignore.

b) Decrypt each $c_{k,j}^s$, i.e., $(d_j, \pi_{d,j}) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c_{k,j}^s, (sid, qid, \mathsf{pk}_{\mathsf{Sig}}^{\mathcal{U}}, \mathsf{pk}_{\mathsf{Sig}}^j))$. If decryption outputs $\perp$, ignore.

c) If $\mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{PTEnc}}^i, d_j, \pi_{d,j}, C_q', (sid, qid)) = \mathsf{false}$ for any $(d_j, \pi_{d,j})$, ignore.

d) If $\mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_j)_{1\le j\le|\mathsf{T}|}, C_q')$ is equal to $1_{\mathbb{G}}$, output $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathsf{true})$. Output $(\mathsf{NOTIFY}, sid, qid, \mathsf{uid}, \mathsf{false})$ otherwise.

Step 12a. Server $\mathcal{T}_i$ continues:

a) On input $(\mathsf{CONTNOTIFY}, sid, qid, \mathsf{uid})$, let $m^{\mathcal{T}_i,\mathcal{S}_j} \leftarrow (sid, qid, \mathcal{S}_j, \mathsf{upk}, time_i', \mathsf{uid})$, if $\mathcal{S}_j \in \mathsf{S}'$ and $m^{\mathcal{T}_i,\mathcal{S}_j} \leftarrow \perp$ otherwise. The values are stored in the record $(\mathsf{PERMISSIONS}, sid, qid, \mathsf{uid}, time_i', \mathsf{S}_i')$. Note, sid, qid and $\mathcal{S}_j$ are not group elements. However, hashing them into $\mathbb{G}_1$ using a collision-resistant hash-function is sufficient, while for uid it is already assumed that it is some group element.

b) If $1_{\mathbb{G}} \neq \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_j)_{j\in[1,n]}, C_q')$, ignore.

c) Sign each $m^{\mathcal{T}_i,\mathcal{S}_j} \neq \perp$, i.e., $\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, m^{\mathcal{T}_i,\mathcal{S}_j})$.

d) Encrypt each value $\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f$, $m^{\mathcal{T}_i,\mathcal{S}_j}$ and s_i, i.e., let $c_i^f \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}^{\mathcal{U}}, ((\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f, m^{\mathcal{T}_i,\mathcal{S}_j})_{j\in[1,n]}, s_i), (sid, qid, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{Sig}}^i)$.

It assumed that each "missing" signature $\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f$ is replaced by an equally long dummy value to account for the different resulting lengths and can be recognized as such.

This is necessary to avoid leaking information due to the different length of the encrypted values.

e) Generate $\sigma_6^i \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (sid, qid, c_i^f, (d_j, \pi_{d,j})_{j\in[1,|\mathsf{T}|]}))$.

f) Send $(sid, qid, c_i^f, (d_j, \pi_{d,j})_{j\in[1,|\mathsf{T}|]}, \sigma_6^i)$ to $\mathcal{U}$.

Step 12b. User $\mathcal{U}$ receives token from $\mathcal{T}_i$:

a) After receiving $(sid, qid, c_i^f, (d_i, \pi_{d,i})_{i\in[1,|\mathsf{T}|]}, \sigma_6^i)$, check σ_6^i. If it is not valid, ignore.

b) Decrypt the received ciphertexts, i.e., let $((\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f, m^{\mathcal{T}_i,\mathcal{S}_j})_{i\in[1,n],j\in[1,|\mathsf{S}|]}, s_i, time_i') \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c_i^f, (sid, qid, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{Sig}}^i, time_i'))$. If the decryption yields $\perp$, ignore.

c) If a different list $(d_i, \pi_{d,i})_{i\in[1,|\mathsf{T}|]}$ was received before, ignore.

d) Check that each signature $\sigma_{\mathcal{T}_i,\mathcal{S}_j}^f$ is valid, and ignore if this is not the case.

e) If $i \neq |\mathsf{T}|$, send $(sid, (\sigma_5^i, (c_{i,j}^s)_{j\in[1,|\mathsf{T}|]})_{i\in[1,|\mathsf{T}|]})$ to $\mathcal{T}_{i+1}$.

f) Otherwise, continue with the next step.

Step 13. User $\mathcal{U}$ receives token:

a) If any $\pi_{d,i}$ is invalid, ignore.

b) If $\mathsf{VfDec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{PTEnc}}^i, d_i, \pi_{d,i}, C_q', (sid, qid)) = \mathsf{false}$ for any $(d_i, \pi_{d,i})$, ignore.

c) If $1_{\mathbb{G}} \neq \mathsf{Dec}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, (d_i)_{1\le i\le|\mathsf{T}|}, C_q')$, ignore.

d) Let $\mathsf{usk}' \leftarrow \mathsf{Combine}_{\mathsf{SS}}(\{s_1, s_2, \ldots, s_n\})$.
e) If $g^{\mathsf{usk}'} \neq \mathsf{upk}$, ignore.
f) Generate the new record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid, \mathcal{U}, \mathsf{uid}, (m^{\mathcal{T}_i,\mathcal{S}_j}, \sigma^f_{\mathcal{T}_i,\mathcal{S}_j})_{i\in[1,|\mathsf{T}|],j\in[1,|\mathsf{S}|]}, time', \mathsf{S}', \mathsf{usk}')$, where $time'$ is the minimum time, i.e., $time' = \min(\bigcup_{i\in[1,|\mathsf{T}|]} time'_i)$, and $\mathsf{S}'' = \{\mathcal{S}_j \mid \forall m^{\mathcal{T}_i,\mathcal{S}_j} : m^{\mathcal{T}_i,\mathcal{S}_j} \neq \perp\}$.
g) Output $(\mathsf{TOKENGENGRANT}, sid, qid, \mathsf{uid}, time'', \mathsf{S}'')$.

Changes for the Protocol to Gain Access to a Service. Now is explained how the protocol used to gain access to a service is altered. Namely, a user additionally uses the reconstructed usk to generate a pseudonym on a string scope and hides the signatures $\sigma^f_{\mathcal{T}_i,\mathcal{S}_j}$ behind a zero-knowledge proof π_s.

Hiding the signatures achieve unlinkability, while neither the complete list of accessible service providers is given away nor the time how long the token used remains valid.

Step 14. Access to Service – User $\mathcal{U}$ sends message to $\mathcal{S}_k$:
a) On input $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, m, \mathcal{S}_k, \Delta, \mathsf{scope})$, get the current time $time$.
b) If there exists a complete record $(\mathsf{tokenreqgranted\text{-}rec}, sid, qid', \mathcal{U}, \mathsf{uid}, (m^{\mathcal{T}_i,\mathcal{S}_k}, \sigma^f_{\mathcal{T}_i,\mathcal{S}_k})_{i\in[1,|\mathsf{T}|]}, (time'_i)_{i\in[1,|\mathsf{T}|],j\in[1,|\mathsf{S}|]}, time', \mathsf{S}', \mathsf{usk})$ with $\mathcal{S}_k \in \mathsf{S}'$, and $time' > time + \Delta$, proceed. Otherwise, ignore. Choose a random record from them.
c) Obtain $\mathsf{pk}_{\mathsf{ENC}}$ for $\mathcal{S}_k$ by sending $(\mathsf{RETRIEVE}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.
d) Let $r' \leftarrow \mathcal{G}(\mathsf{usk}, \mathsf{scope}, \mathcal{S}_k)$, where $\mathcal{G} : \mathbb{Z}_q \times \{0,1\}^* \times \{0,1\}^* \to \mathbb{Z}_q$ is a random oracle, and $\mathsf{nym} \leftarrow g_c^{\mathsf{usk}} h_c^{r'}$.
e) Generate the following proof:

$$\pi_s \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{((\sigma^f_{\mathcal{T}_j,\mathcal{S}_k}, m_2^{\mathcal{T}_j,\mathcal{S}_k}, m_4^{\mathcal{T}_j,\mathcal{S}_k}, m_5^{\mathcal{T}_j,\mathcal{S}_k}, m_6^{\mathcal{T}_j,\mathcal{S}_k})_{j\in[1,|\mathsf{T}|]}, \mathsf{usk}, r') :$$
$$\bigwedge_{j\in[1,|\mathsf{T}|]} \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}^j_{\mathsf{Sig}}, m^{\mathcal{T}_j,\mathcal{S}_k}, \sigma^f_{\mathcal{T}_j,\mathcal{S}_k}) = \mathsf{true} \ \wedge \ \bigwedge_{j\in[1,|\mathsf{T}|]} m_5^{\mathcal{T}_j,\mathcal{S}_k} \geq time + \Delta \ \wedge$$
$$\bigwedge_{j\in[1,|\mathsf{T}|-1]} m_2^{\mathcal{T}_j,\mathcal{S}_k} = m_2^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \ \wedge \ \bigwedge_{j\in[1,|\mathsf{T}|-1]} m_6^{\mathcal{T}_j,\mathcal{S}_k} = m_6^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \ \wedge$$
$$m_4^{\mathcal{T}_1,\mathcal{S}_k} = g^{\mathsf{usk}} \wedge \bigwedge_{j\in[1,|\mathsf{T}|-1]} m_4^{\mathcal{T}_j,\mathcal{S}_k} = m_4^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \ \wedge \ \mathsf{nym} = g_c^{\mathsf{usk}} h_c^{r'}\}(n'')$$

where $n'' = (sid, qid, \mathcal{S}_k, time)$.
f) Encrypt m, i.e., let $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, (m, time, \Delta, \mathsf{nym}, \mathsf{scope}, \pi_s), (sid, qid))$.
g) Send (sid, qid, c) to $\mathcal{S}_k$.

Step 15. Access to Service – Service $\mathcal{S}$ received message from $\mathcal{U}$:
a) On input (sid, qid, c), ask $\mathcal{F}_{\mathsf{CRS}}$ to obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \mathsf{pp}_{\mathsf{NYM}}, \mathsf{pp}_{\mathsf{SS}}, \mathsf{pp}_{\mathsf{TEnc}})$ by sending (GETCRS, sid) to $\mathcal{F}_{\mathsf{CRS}}$.
b) If there is no record $(\mathsf{KEYPAIRSSOSERVICE}, (\mathsf{sk}^{\mathcal{S}_k}_{\mathsf{ENC}}, \mathsf{pk}^{\mathcal{S}_k}_{\mathsf{ENC}}))$, ignore.
c) Obtain $(\mathsf{pk}^i_{\mathsf{ENC}}, \mathsf{pk}^i_{\mathsf{Sig}})$ for each server $\mathcal{T}_i \in \mathsf{T}$ by sending $(\mathsf{RETRIEVE}, (\mathcal{T}_i, (sid, \mathcal{F}_{\mathsf{CA}})))$ to $\mathcal{F}_{\mathsf{CA}}$.
d) Decrypt c, i.e., let $(m, time, \Delta, \mathsf{nym}, \mathsf{scope}, \pi_s) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}^{\mathcal{S}}_{\mathsf{ENC}}, c, (sid, qid))$. If decryption fails, ignore.

e) Verify π_s. If it is not valid, ignore.
f) Get the current time $time'$. If $time' > time + \Delta$, ignore.
g) Let $\mathsf{nym}' \leftarrow \mathsf{Eval}_{\mathsf{PRF}}(\kappa, \mathsf{nym})$, where κ is taken from record $(\mathsf{KEYPAIRSSOSERVICE}, (\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}), \kappa)$.
h) Output $(\mathsf{RECVMESSAGE}, sid, qid, \mathsf{nym}', m, \mathsf{scope})$.

Note, only the signatures $\sigma^f_{i,\mathcal{S}_j}$ need to support efficient zero-knowledge proofs, i.e., all other signatures can still be realized with more a efficient standard signature scheme such as RSA-FDH. However, to have a compact representation, this is not made explicit.

Theorem 3.11. *The protocol described above securely implements the ideal functionality $\mathcal{F}^2_{\mathsf{sso}}$ in the $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid model, if* DSIG*,* ENC*,* TEnc*,* SS*,* NIZKPoK *and* NYM *are secure, assuming static corruptions.*

3.3.3 Security of Protocol 2

High-Level Idea. As for the simpler protocol, the proof of Theorem 3.11 is done by providing a simulator and a sequence of games. The initial game is the real experiment and the final game (Game 12) runs the simulator given only the information that is also available to the adversary when interacting with the ideal functionality.

As for the first protocol, the proof idea is as follows. In the first game, the real-world is executed, where a challenger still receives all inputs and outputs from the honest parties. With each game-hop, the protocol is gradually made less depended on the input received for the honest party and prove why this does not change the view of the adversary in a non-negligible way. Eventually, in the final hop, the protocol can be simulated with the given simulator, barely based on the information it receives from the ideal functionality which proves security.

However, as the hops and simulator SIM are very similar, only the differences are explained in more detail.

Sequence of Games. The sequence of games is essentially the same as for the proof given in Section 3.2.3 for the first protocol, with the exception of the following alterations. First, an additional game hop is needed:

Game 10+1: In this hop, the challenger replaces the c^f_i from an honest ticket-granting server encrypted for a honest user by encryptions of 1. As usual, a standard hybrid argument shows, that this hop is (so far) indistinguishable due to the IND-CCA2 security of the encryption scheme.

Note, replacing (by the adversary) is not possible, as the shares are signed and thus forgeries are already excluded. Moreover, they cannot be re-used in different contexts, due to the attached label, while, also due to the attached label, any ciphertexts provided by the adversary can simply be decrypted using the decryption oracle provided.

Moreover, Game 11 needs to be changed as follows.

Game 11: The challenger now aborts, if a service outputs a message from a corrupt user, even though the user in question never received a (valid) token from the ticket-granting servers,

if at least one ticket-granting server is honest. As this implies that there is at least one σ_j^f for a message m_j for which the ticket-granting servers have never issued a signature (note, honest users do not receive any signatures), this tuple breaks the unforgeability of the signature scheme.

A reduction is simple, using a hybrid argument: $\mathsf{pk}_{\mathsf{Sig}}$ is embedded as one of the ticket-granting servers' one. Then, each signature is generated using the the signature-generation oracle provided. This does not change the view of the adversary so far. Moreover, both the message and signature can easily be extracted from π_s. Then, if the above case happens, the challenger wins its own unforgeability game with non-negligible probability due to the random embedding.

Finally, some additional hops are required to account for the pseudonyms incorporated into the protocol.

Game 11+1: The challenger now takes control of $\mathcal{G}$ and (lazy) simulates it honestly. Clearly, this does not change the view of the adversary.

Game 11+2: The challenger now aborts, if it draws a key pair usk of the pseudonym system twice for honest parties. Once more, this can only happen with negligible probability due to the exponential key-space.

Game 11+3: The challenger now aborts, if it draws $1_{\mathbb{G}}$ as a response for $\mathcal{G}$. This only happens with negligible probability due to the exponential size of $\mathbb{G}$.

Game 11+4: The challenger now aborts, if the adversary makes a query $(\mathsf{usk}, \cdot, \cdot)$ to $\mathcal{G}$ without having any successful token generation run for an honest user with uid which belongs to that usk, or it guesses such a output correctly. Note, there is at most one secret key usk in the pseudonym-system for each upk. This also includes the case where r is chosen randomly (without querying $\mathcal{G}$); then, the challenger randomly assigns the values as a valid pre-image to $\mathcal{G}$ (if it does not exist yet).

Clearly, due to the random choice of usk and the outputs from sets with exponential size, this can only happen with negligible probability.

Game 11+5: The challenger now aborts, if a value in the random oracle $\mathcal{G}$ was drawn twice. Due to the birthday bound, this can only happen with negligible probability.

Game 11+6: Next is changed how g_c and h_c are calculated. In particular, the challenger draws a random $g_c \xleftarrow{\$} \mathbb{G}^\times$ and a random $x \xleftarrow{\$} \mathbb{Z}_q^*$ and embeds (g_c, g_c^x) as the new public parameters for $\mathsf{pp}_{\mathsf{NYM}}$. This is only an internal change, as the real CRS looks just alike, but the challenger now knows x.

Game 11+7: In this game hop is shown that the adversary cannot find collisions, if different usks are involved. Assume, towards contradiction, that the adversary was able to find a collision $\mathsf{nym}' = \mathsf{nym}$, while $\mathsf{usk} \neq \mathsf{usk}'$ (Note, either each usk is chosen honestly, one can be extracted from π_s or even both). One can then extract a solution to a DL-challenge. In particular, the reduction receives $(\mathbb{G}, g, q, g^x)$, and embeds g and g^x as $\mathsf{pp}_{\mathsf{NYM}}$. Now, as,

by assumption, it holds that $g^{\mathsf{usk}} g^{xr_1} = g^{\mathsf{usk}'} g^{xr_2}$ (enforced by π_s and calculated honestly by the challenger), where thus $r_1 \neq r_2$ must hold. Hence, x can be extracted by calculating $\mathsf{usk} - \mathsf{usk}'/r_2 - r_1$ and can then directly be returned.

Game 11+8: Now is changed how pseudonyms are calculated in case not all ticket-granting servers are corrupt and the that uid has neither been created by the adversary nor the adversary received usk, i.e., if it guessed a password correctly and thus received usk. Namely, instead of computing $r' \leftarrow \mathcal{G}(\mathsf{usk}, \mathsf{scope}, \mathcal{S}_k)$ and $\mathsf{nym} \leftarrow g_c^{\mathsf{usk}} h_c^{r'}$, choose a random $r' \xleftarrow{\$} \mathbb{Z}_q^*$ for each new scope and $\mathcal{S}_k$ combination (and abort if r' was drawn twice. However, due to the birthday bound this changes the view of the adversary only negligibly.) Once, however, the adversary gets to know usk, i.e., if $\mathcal{A}$ (successfully) logs in into an honestly generated account, and thus expects to receive usk, the challenger proceeds as follows. It programs $\mathcal{G}$ such that for all prior (unique) $(\mathsf{usk}, \mathsf{scope}, \mathcal{S}_k)$ combinations it returns $(r - \mathsf{usk})/x$. This programming does not change the view of the adversary. Note, usk is still distributed as before.

Game 11+9: The challenger now replaces each generated nym' by the PRF from the honest services with a random value and keeps the output consistent. A distinguisher can be turned against the pseudo-randomness of the used PRF. The reduction works as follows. Using a hybrid-argument, the challenger replaces the PRFs one service provider at a time, and uses the provided oracle for generation of nym'. Whatever the adversary outputs, is also output by the reduction.

Game 11+10: The challenger now aborts, if it draws a randomly drawn nym' twice. Clearly, this only happens with negligible probability due to the birthday bound.

Game 11+11: The challenger now aborts, if not all ticket-granting servers are corrupt, but a service receives an already seen nym' from a different upk but the same string scope. Clearly, this breaks the collision-resistance of the pseudonym-system. A reduction is straightforward. Namely, usk can be extracted from π_s in the case of a corrupt user, and are known for an honest user anyway. The scopes are known in all cases, either by decryption or as input by the environment. Note, the challenger already aborts in the case of forged proofs, and all proofs by honest parties are simulated. Thus, all values can be extracted and can simply be returned to the challenger, as the $\mathsf{usk}_0 \neq \mathsf{usk}_1$ was already excluded.

Game 11+12: The challenger now aborts, if an honest service outputs a message from a corrupt user along with nym', even though that upk has never been issued a currently valid token, and at least one ticket-granting server is honest. As this implies that there is at least one σ_j^f for a message m_j for which the ticket-granting servers have never issued a signature (note, honest users do not receive any signatures from honest servers in this hop already), this tuple breaks the unforgeability of the signature scheme. A reduction is simple: $\mathsf{pk}_{\mathsf{Sig}}$ is embedded as one of the ticket-granting servers' one. Then, each signature is generated using the the signature-generation oracle provided. This does not change the view of the adversary so far. Moreover, both values can simply be extracted from π_s. Then, if the above case happens, the

challenger wins its own unforgeability game with non-negligible probability due to the random embedding.

These are all and additions changes needed.

3.3.3.1 Simulator

Now, the simulator for the second protocol is presented. The simulator essentially is the same as for the first protocol, only with a few small alterations.

Namely, only the token-generation and the simulation for the messaging part have to be altered as follows:

3.3.3.2 Simulation of the Token-Generation Protocol

The only change required is the following. As soon as the adversary was able to receive a token for an account uid which belongs to a simulated user $\mathcal{U}$, i.e., before the simulator SIM sends the final network message to $\mathcal{A}$ and thus has still control, it programs the random oracle $\mathcal{G}$ as described in the game hops. The required scopes are learned by sending (CORRSCOPES, *sid*, uid) to the ideal functionality.

Note, this step is only necessary, if there is at least one honest ticket-granting server.

3.3.3.3 Simulation of the Messaging Protocol

Honest User $\mathcal{U}$. First, the case of an honest user is described.

$\mathcal{S}$ *is honest.* Here, the simulator needs to branch, depending on whether all ticket-granting servers are corrupt or not and whether the adversary interfered, if it holds a valid token for the uid in question (and thus knows usk).

Not all ticket-granting servers are corrupt. The simulator is woken up when it receives (SENDMESSAGE, *sid*, *qid*, $\mathcal{U}, \mathcal{S}$). In the case that not all ticket-granting servers are corrupt the adversary cannot tamper with the reconstruction of usk. Thus, the simulator simply sends an empty ciphertext to the simulated service.

If the adversary did not interfere and thus the service receives the sent ciphertext, the simulator triggers the output in the real world by sending (RECVMESSAGE, *sid*, *qid*, $\perp$) to the ideal functionality. Note, the ideal functionality "magically" chooses the correct pseudonym, as it is guaranteed that usk was restored correctly, even in the case where a corrupt user sends a message before the corresponding records are generated, as the ideal functionality takes care of this case.

If the adversary interfered, however, i.e., if the adversary holds a valid token for uid and $\mathcal{S}$, while the simulator receives (sid, qid, c') (which makes the simulated service $\mathcal{S}$ output some message m', i.e., the adversary overruled the request), the simulator needs to reflect that behavior in the ideal world as well. Namely, the simulator proceeds as in the case for a corrupt user, described below.

All ticket-granting servers are corrupt. In this case, the simulator also receives (SENDMESSAGE, sid, qid,$\mathcal{U}$,$\mathcal{S}$). However, the adversary is now able to exchange usk at will. Then, if the adversary did not interfere, it sends (RECVMESSAGE, sid, qid, usk) to the ideal functionality, triggering the output in the real world with the correct pseudonyms. Note, an honest user will always calculate such pseudonyms correctly, while the adversary may decide to give that usk to a different user as well (or even use it itself, as described now).

If the adversary interfered, however, i.e., if the adversary holds a valid token for uid and $\mathcal{S}$, while the simulator receives (sid, qid, c') (which makes the simulated service $\mathcal{S}$ output some message m', i.e., the adversary overruled the request), the simulator needs to reflect that behavior in the ideal world as well. Namely, the simulator proceeds as in the case for a corrupt user, described below.

$\mathcal{S}$ is corrupt. If the service provider is corrupt, the simulator also needs to branch, depending on whether the adversary knows usk (but at least one ticket-granting server remains honest) or if all ticket-granting servers are corrupt. Note, the adversary $\mathcal{A}$ knows usk if either the registration was done by a corrupt user or the adversary was able to successfully receive a token for an account which was created by a honest user, i.e., guessed the password correctly.

All ticket-granting servers are corrupt. Here, the simulator is woken up upon reception of (SENDMESSAGE, sid, qid, uid,$\mathcal{U}$,$\mathcal{S}$,m,Δ, scope). It then simply follows the protocol, as all values are known, while the proofs are simulated.

Note, triggering any output in the ideal world is not necessary.

Not all ticket-granting servers are corrupt, but $\mathcal{A}$ knows usk. This is the same simulation as for the case that all ticket-granting servers are corrupt. However, in this case the pseudonyms are already programmed correctly and thus the simulator can proceed without any additional alterations.

Note, triggering any output in the ideal world is not necessary.

Not all ticket-granting servers are corrupt and $\mathcal{A}$ does not know usk. Here, the simulator SIM is woken up with (SENDMESSAGE, sid, qid, L,$\mathcal{U}$,$\mathcal{S}$,m,Δ, scope), where L is the list of all qid's with the same uid, $\mathcal{S}$ and scope combination. If that list is empty, the simulator chooses a new random (and not yet chosen) pseudonym nym (as described in the game-hops) and then simply continues the protocol, also storing (qid', nym). If, however, the list L is not empty, the simulator uses the pseudonym nym from the record (qid', nym). Note, once usk becomes known to the adversary, all pseudonyms are programmed as described in the sequence of games and at the simulation of the token-generation protocol. The case of an adversary holding a valid token for uid cannot happen, as otherwise usk would be known, which is already excluded.

Note, triggering any output in the ideal world is not necessary.

Corrupt User $\mathcal{U}$. Now, the case of a corrupt user is described. In this case, the simulated service $\mathcal{S}$, i.e., the simulator SIM, receives (sid, qid, c') on the network.

$\mathcal{S}$ is honest. Again, the simulator needs to branch, depending on whether all ticket-granting servers are corrupt or not. If the simulated $\mathcal{S}$ would output some message m' (and some nym'), the simulator behaves as follows.

Not all ticket-granting servers are corrupt. In this case, the simulator mimics the output in the ideal world by sending $(\mathsf{SENDMESSAGE}, sid, qid, \mathsf{uid}, m', \mathcal{S}, \Delta', \mathsf{scope}')$ to the ideal functionality. The required values can simply be extracted from π_s and by decrypting c'. The simulator SIM directly receives back control and now behaves as follows. If the pseudonym was generated honestly for that uid (this can simply be checked by looking at the responses/queries by $\mathcal{G}$ and π_s), the simulator sends $(\mathsf{RECVMESSAGE}, sid, qid, \perp)$ to the functionality. In the case the pseudonym NYM was not correctly calculated (as defined in the protocol), the simulator sends $(\mathsf{RECVMESSAGE}, sid, qid, \mathsf{NYM})$ to the ideal functionality. Note, collisions in the space of NYM is already ruled out for different usks.

All ticket-granting servers are corrupt. In this case, the adversary can exchange the pseudonym secret key at will. This simulation proceeds as in the prior case. However, in this case uid no longer matters (and is hidden from the service anyway), as the ideal functionality explicitly ignores this case.

$\mathcal{S}$ is corrupt. This is completely internal to the adversary $\mathcal{A}$, and thus no simulation is necessary.

3.4 Concrete Zero-Knowledge Proofs

Now is presented how the zero-knowledge proofs can be constructed. For the sake of completeness, it is assumed that the $\mathsf{IND\text{-}CPA}$ secure encryption scheme is ElGamal and the TEnc the one given in Appendix A.

Assume the system parameters contain a group $\mathbb{G} = \langle g \rangle$ of prime order $q = \Theta(\lambda)$ that is also used for TEnc, i.e., let $\mathcal{F}_{\mathsf{CRS}}$ return an element $\tilde{y} \in \mathbb{G}$ and let $(\mathbb{G}, g, q)$ be the input $\mathsf{KeyGen}_{\mathsf{TEnc}}$ so that the values $\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pk}_{\mathsf{PTEnc}}$ output by $\mathsf{KeyGen}_{\mathsf{TEnc}}$ are elements of $\mathbb{G}$. In this discrete-logarithm-based setting, the various proofs used in the protocols can be instantiated using so-called generalized Schnorr-protocols [CKY09, Sch91].

When referring to such proof protocols, the following notation, similar to what was used in the high-level description, is used [CKY09, CS97]. For instance,

$$\mathsf{SPK}\Big[(a, b, c) : y = g^a h^b \wedge \tilde{y} = g^a h^c\Big](m)$$

denotes a "*Signature based on a zero-knowledge Proof of Knowledge of integers a, b, c such that $y = g^a h^b$ and $\tilde{y} = g^a h^c$ holds*," where y, g, h, and $\tilde{y}$ are elements of $\mathbb{G}$ and where m is included into the hash that is used to make the proof of knowledge protocol non-interactive (Fiat-Shamir transformation).

The convention is that the letters in the parenthesis (a, b, c) denote quantities of which knowledge is being proven, while all other values are known to the verifier.

Given a protocol in this notation, it is straightforward to derive an actual protocol implementing the proof. Indeed, the computational complexities of the proof protocol can be easily derived from this notation: basically for each term $y = g^a h^b$, the prover and the verifier have to perform an equivalent computation, and to transmit one group element and one response value for each exponent. Refer to, e.g., Camenisch et al. [CKY09] for details on this.

The most efficient way to make these protocol concurrent zero-knowledge, simulation-sound, and non-malleable is by the Fiat-Shamir transformation [FS86]. In this case, one has to resort to the random-oracle model [BR93] for the security proof.

To make the resulting non-interactive proofs simulation-sound in the required context, it suffices to let the prover include context information as an argument to the random oracle in the Fiat-Shamir transformation, such as the system parameters, uid, qid, and the protocol step in which the statement is being proven, and a collision-resistant hash of the communication transcript that the prover and verifier have engaged in so far, so that the proof is resistant to a man-in-the-middle attack [BPW12a, FKMV12].

In particular, notice that all the statements parties prove to each other in the concrete proofs are only proofs of membership (i.e., that some computation was done correctly) and *not* online-extractable proofs of knowledge, as the values which need to be extracted are "encrypted to the sky". Therefore, it is not necessary that the prover can be re-wound to extract the required witnesses in the simulator.

Summarized, all proofs are Σ-protocols [Sch91], made non-interactive using, e.g., the Fiat-Shamir transform [FS86], forming a proof of membership which do not require rewinding. Moreover, these proofs are simulation-sound, as context-information is attached as a label - which, in the case of Fiat-Shamir, corresponds to put the label into the hash-function as well [FKMV12].

To this end, assume that an element $\tilde{y}$ is available as part of the CRS that is used as a public key for ElGamal encryptions in the protocols, i.e., a public key "in the sky". The simulator then sets $\tilde{y}$ so that it knows $\log_g \tilde{y}$ and hence can, by decryption, extract group elements (the password and the password attempts in this case) without rewinding.

The Proof π_0. The proof can be constructed as follows, where $\pi_0 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\overline{\mathsf{pwd}}, r_p) : C_p = \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}; r_p)\}(n)$, where $n = (sid, qid, \mathsf{uid}, \mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pk}_{\mathsf{PTEnc}}, C_p)$. The ciphertext itself is composed as $C_p = (C_{p,1}, C_{p,2}) = (g^{r_p}, \mathsf{pwd} \cdot \mathsf{pk}_{\mathsf{TEnc}}^{r_p})$.

The proof π_0 as now realized as follows.

First, the prover computes the value $\tilde{C}_{p,2} = \mathsf{pwd} \cdot \tilde{y}^{r_p}$ that, together with $C_{p,1}$, forms an encryption of pwd under $\tilde{y}$. Then it follows:

$$\pi_0 \xleftarrow{\$} (\tilde{C}_{p,2}, \mathsf{SPK}\left[(r_p) : C_{p,1} = g^{r_p} \wedge C_{p,2}/\tilde{C}_{p,2} = (\mathsf{pk}_{\mathsf{TEnc}}/\tilde{y})^{r_p}\right](n, \tilde{C}_{p,2}))$$

The Proof π_1. The proof is given as $\pi_1 \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(C_{p'}, \overline{\mathsf{pwd}'}, r'_{\mathsf{pwd}'}, r_q) : C_q = (C_p \odot C_{p'})^{r_q} \wedge C_{p'} \in \mathsf{Enc}_{\mathsf{TEnc}}(\mathsf{pk}_{\mathsf{TEnc}}, \mathsf{pwd}'^{-1})\}(n')$, where $n' = (n, qid, C_q, \mathsf{pk}_{\mathsf{ENC}}^{\mathrm{CCA2}})$.

First, the prover computes the values $\tilde{C}_{p'} = (\tilde{C}_{p',1} \leftarrow g^{r'_p}, \tilde{C}_{p',2} \leftarrow \mathsf{pwd}' \cdot \tilde{y}^{r'_p})$ that form an encryption of pwd' under $\tilde{y}$ (i.e., the public key contained in the CRS). Next, the prover computes $C_q = (C_p \odot C_{p'})^{r_q} \leftarrow (C_{p,1}^{r_q} g^{\bar{r}}, (C_{p,2}/\mathsf{pwd}')^{r_q} \mathsf{pk}_{\mathsf{TEnc}}^{\bar{r}})$.

Then, the proof can be realized by:

$$\pi_1 \xleftarrow{\$} ((\tilde{C}_{p',1}, \tilde{C}_{p',2}), \mathsf{SPK}\Big[(r_q, \bar{r}, r_{p'}, r^*) : C_{q,1} = C_{p,1}{}^{r_q} g^{\bar{r}} \wedge$$
$$C_{q,2} = (\frac{C_{p,2}}{\tilde{C}_{p',2}})^{r_q} \tilde{y}^{r^*} \mathsf{pk}_{\mathsf{TEnc}}^{\bar{r}} \wedge \tilde{C}_{p',1} = g^{r'_p} \wedge 1 = \tilde{C}_{p',1}^{r_q} (1/g)^{r^*}\Big](n', (\tilde{C}_{p',1}, \tilde{C}_{p',2}))),$$

where $r^* = r_q r'_p$. Why this proof works is discussed next. The last two terms establish that $r^* = r_q r'_p$ holds and thus one can rewrite the second term as $C_{q,2} = (\frac{C_{p,2}}{\tilde{C}_{p',2}})^{r_q} \tilde{y}^{r_q r'_p} \mathsf{pk}_{\mathsf{TEnc}}^{\bar{r}} = (\frac{C_{p,2}}{\tilde{C}_{p',2}/\tilde{y}^{r'_p}})^{r_q} \mathsf{pk}_{\mathsf{TEnc}}^{\bar{r}}$. Now, as $\tilde{C}_{p',2}/\tilde{y}^{r'_p}$ equals the plaintext encrypted under $\tilde{y}$, say pwd' it follows that $C_{q,2} = (\frac{C_{p,2}}{\mathsf{pwd}'})^{r_q} \mathsf{pk}_{\mathsf{TEnc}}^{\bar{r}}$. Assuming that $C_p = (g^{r_p}, \mathsf{pwd} \cdot \tilde{y}^{r_p})$ for some r_p and pwd, it holds that $C_q = (g^{r_p r_q + \bar{r}}, (\mathsf{pwd}/\mathsf{pwd}')^{r_q} \mathsf{pk}_{\mathsf{TEnc}}^{r_p r_q + \bar{r}})$, i.e., that C_q is a properly randomized encryption of the password quotient under $\mathsf{pk}_{\mathsf{TEnc}}$.

The Proofs $\pi_{2,i}$. The proof $\pi_{2,i} \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r_i^1, r_i^2) : C_{q,i} = (g^{r_i^1} C_1^{r_i^2}, y^{r_i^1} C_2^{r_i^2})\}(n')$ can be constructed as follows, where $C_q = (C_1, C_2)$, and $C_{q,i} = (C_3, C_4)$, where n' is as in the prior proof, while y is the public key of the TEnc:

$$\pi_{2,i} \xleftarrow{\$} \mathsf{SPK}\Big[(r_i^1, r_i^2) : C_3 = g^{r_i^1} C_1^{r_i^2} \wedge C_4 = y^{r_i^1} C_2^{r_i^2}\Big](n')$$

The Proof π_s in $\mathcal{F}_{\mathsf{sso}}^2$. The second protocol requires a rather involved proof π_s, where *all* values need to be online-extractable:

$$\pi_s \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{((\sigma^f_{\mathcal{T}_j,\mathcal{S}_k}, m_2^{\mathcal{T}_j,\mathcal{S}_k}, m_4^{\mathcal{T}_j,\mathcal{S}_k}, m_5^{\mathcal{T}_j,\mathcal{S}_k}, m_6^{\mathcal{T}_j,\mathcal{S}_k})_{j\in[1,|\mathsf{T}|]}, \mathsf{usk}, r') :$$
$$\bigwedge_{j\in[1,|\mathsf{T}|]} \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig}}^j, m^{\mathcal{T}_j,\mathcal{S}_k}, \sigma^f_{\mathcal{T}_j,\mathcal{S}_k}) = \mathsf{true} \wedge \bigwedge_{j\in[1,|\mathsf{T}|]} m_5^{\mathcal{T}_j,\mathcal{S}_k} \geq time + \Delta \wedge$$
$$\bigwedge_{j\in[1,|\mathsf{T}|-1]} m_2^{\mathcal{T}_j,\mathcal{S}_k} = m_2^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \wedge \bigwedge_{j\in[1,|\mathsf{T}|-1]} m_6^{\mathcal{T}_j,\mathcal{S}_k} = m_6^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \wedge$$
$$m_4^{\mathcal{T}_1,\mathcal{S}_k} = g^{\mathsf{usk}} \wedge \bigwedge_{j\in[1,|\mathsf{T}|-1]} m_4^{\mathcal{T}_j,\mathcal{S}_k} = m_4^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k} \wedge \mathsf{nym} = g_c^{\mathsf{usk}} h_c^{r'}\}(n'')$$

Now is discussed how this proof can be efficiently realized. The main idea is to use a recent signature scheme by Groth for the token-generation servers. This scheme allows one to sign group elements and to efficiently prove knowledge of a signature and messages where one can use the ElGamal encryption scheme to make the latter proof efficiently online-extractable. Groth's scheme is recalled in Appendix B. It is also explained how to prove knowledge of a signature on messages that are encrypted under a public key given in the CRS. This allows one to implement the first terms in the proof π_s. The terms $\bigwedge_{j\in[1,|\mathsf{T}|-1]} m_2^{\mathcal{T}_j,\mathcal{S}_k} = m_2^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k}$, $\bigwedge_{j\in[1,|\mathsf{T}|-1]} m_4^{\mathcal{T}_j,\mathcal{S}_k} = m_4^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k}$ and $\bigwedge_{j\in[1,|\mathsf{T}|-1]} m_6^{\mathcal{T}_j,\mathcal{S}_k} = m_6^{\mathcal{T}_{|\mathsf{T}|},\mathcal{S}_k}$ can simply be provided by

encrypting qid'_j, uid, and upk, respectively, only once and then use the same ciphertext in the proof terms when proving knowledge of a signature (cf. Appendix B). It remains to discuss the term $\bigwedge_{j\in[1,|\mathsf{T}|]} time'_j \geq time + \Delta$. There are well known methods to implement range proofs. However, the most efficient ones require an RSA modulus being available as part of the CRS which is hard to achieve without a trusted party generating the CRS. In this case, a more efficient approach is as follows, assuming that tokens are valid for a limited time (e.g., a day) and that the granularity is not too fine (e.g., a minute). Then there are relatively few values of Δ that are allowed (e.g, $1'440 = 24 * 60$) and one can make available a Groth signature of $g_1^{\Delta_i}$ for all possible values. The expiration time $time'_j$ of a token can then be encoded as a value in $\mathbb{Z}_q$ and then $m = g_1^{time'_j}$ be signed as part of the CRS. One can then prove that $time'_j \geq time + \Delta$ by proving knowledge of a signature on two messages m_1 and m_2 s.t. $g_1^{time} = m_1 \cdot m_2$, which translates into proving knowledge of the randomness used for encryption $\bar{r}$ s.t. $g_1^{time}/(c_1 \cdot c_2) = (\tilde{y}_1\tilde{y}_2)^{\bar{r}}$, where c_1 and c_2 are the encryptions of m_1 and m_2 under $\tilde{y}_1$ and $\tilde{y}_2$, respectively, that are used to prove knowledge of a signature on m_1 and m_2, as presented in Appendix B. Finally, usk and r' can be made online-extractable using, e.g., the encryption scheme by Camenisch and Shoup [CS03] or Paillier [Pai99].

The Proofs $\pi_{d,i}$. The concrete proofs $\pi_{d,i}$ for the threshold encrypion scheme TEnc are presented in Appendix A.

3.5 Efficiency of the Protocols

This section contains a short summary of a prototypical implementation of both schemes. A more detailed description is given in Appendix E.

The implementation was done using Java 9 without any optimization, while only a single thread does the calculations. All entities were calculated on a standard PC with a 2.66GHz processor and 8GiB RAM. To instantiate the random oracle and the PRF, SHA-512 (properly truncated for the challenges in the NIZKs) was used. Each ticket-granting server always allows to proceed, while it allows access to 10% of the services for ten time units. The group $\mathbb{G}$ used in the first protocol was $\mathbb{G}_1$ of the "SNARK_2" curve, as given in the pairing library provided by the TU Graz.[2] The zero-knowledge proofs are the ones presented in Section 3.4. The exponents usk and r' are made online-extractable using the encryption scheme by Camenisch and Shoup [CS03] with 2,048 Bit moduli. The ideal functionality $\mathcal{F}_{\mathsf{CA}}$ was realized using hard-coded keys, while all signatures without ZK-property are standard ECDSA. For the range-proof in π_s, 1,440 signatures are generated in the CRS, which corresponds to a maximum validity of one day, if the granularity is one second. For secret-sharing of usk in the second protocol, Shamir's protocol was used [Sha79].

To have a meaningful evaluation, both protocols where measured 100 times with $|\mathsf{T}| \in \{3, 5, 10\}$ and $|\mathsf{S}| \in \{100, 500, 1'000\}$, which even covers very large environments. Moreover, the numbers for network delay and generating the *qid*s are not part of the measurements, as it depends on the concrete setting of a potential deployment. Finally, the timings for the ticket-granting servers are accumulated for token-generation and key-generation, as this is

[2] `https://jce.iaik.tugraz.at`

Table 3.1: Overview of the measurements for the first protocol. All values are in milliseconds.

$\lvert\mathsf{T}\rvert$		3			5			10		
$\lvert\mathsf{S}\rvert$		100	500	1000	100	500	1000	100	500	1000
ParGen	Avg.	2	2	2	2	2	2	2	2	2
	Med.	1	2	2	2	2	2	1	2	2
KGen ($\mathcal{T}$)	Avg.	34	24	24	42	43	40	82	87	81
	Med.	24	24	24	40	42	39	82	84	80
KGen ($\mathcal{S}$)	Avg.	636	3′171	6′450	656	3′310	6′245	641	3′346	6′362
	Med.	628	3′134	6′364	636	3′293	6′223	640	3′336	6′344
Reg ($\mathcal{U}$)	Avg.	60	58	58	97	98	94	174	183	184
	Med.	59	57	58	95	98	93	174	182	183
Reg ($\mathcal{T}$)	Avg.	144	135	140	526	542	529	3′276	3′509	3′767
	Med.	143	134	138	518	545	527	3′278	3′478	3′765
TGen ($\mathcal{U}$)	Avg.	107	101	103	169	173	165	288	307	315
	Med.	106	100	102	164	172	165	288	303	315
TGen ($\mathcal{T}$)	Avg.	647	639	654	1′716	1′758	1′663	6′168	6′550	6′369
	Med.	634	634	649	1′674	1′742	1′657	6′159	6′517	6′350
Comm ($\mathcal{U}$)	Avg.	11	11	11	11	12	11	11	12	12
	Med.	10	11	11	11	11	11	11	12	12
Comm ($\mathcal{S}$)	Avg.	22	20	21	29	30	29	42	45	48
	Med.	21	20	20	28	30	29	42	44	48

what a user (or the administrator setting up the servers resp.) sees. This was done, as UC requires that at most one party has control. It is stressed, however, that key-generation can be parallelized, as they the keys do not depend on any prior state. The values for each server can thus be calculated by dividing it by $|\mathsf{T}|$ (or $|\mathsf{S}|$ resp.), as each ticket-granting server has to perform the exact same amount of work. The only exception is the communication with a single service, as only one service and the user is involved.

The measurements for the first protocol are in Table 3.1, while the results for the second protocol are depicted in Table 3.2.

As expected, the parameter-generation is independent of $|\mathsf{S}|$ and $|\mathsf{T}|$, while key-generation obviously only depends on the number of servers involved for both protocols. The same is true for the registration part of both protocols, which only depends on the amount of ticket-granting servers $|\mathsf{T}|$.

Things slightly change, however, for token-generation. In the first protocol, the numbers are more or less constant, as each ticket-granting server only signs a single bit-string, regardless of how many services are in the system. More precisely, this string's length depends on $|\mathsf{S}|$, but the signing operation is much more costly and hides this operation. This is, however, not true for the second protocol, where each ticket-granting server needs to generate a signature for each service a user can access. Thus, this overhead is now very visible. Likewise, at "token-presentation", i.e., communication with a service, the user and service either only need to generate and verify some signatures in the first protocol, while for the second protocol a rather expensive NIZK needs to be generated which hides the signatures generated by *each* ticket-granting server.

Summarized, both protocols can be considered practical, especially taking into account that parameter generation, registration and token-generation are rather infrequent (to some extend

Table 3.2: Overview of the measurements for the second protocol. All values are in milliseconds.

$\lvert\mathsf{T}\rvert$		3			5			10		
$\lvert\mathsf{S}\rvert$		100	500	1000	100	500	1000	100	500	1000
ParGen	Avg.	28′705	28′066	29′630	27′433	28′965	29′854	32′719	30′057	28′727
	Med.	26′958	26′234	28′131	25′552	28′001	28′593	31′683	29′085	26′718
KGen ($\mathcal{T}$)	Avg.	83	83	85	140	139	140	313	280	280
	Med.	82	83	84	139	139	139	318	277	279
KGen ($\mathcal{S}$)	Avg.	622	3′146	6′403	639	3′140	6′310	697	3′164	6′271
	Med.	619	3′127	6′360	638	3′113	6′292	713	3′127	6′237
Reg ($\mathcal{U}$)	Avg.	57	58	61	92	92	94	195	179	180
	Med.	56	58	60	91	92	93	196	176	179
Reg ($\mathcal{T}$)	Avg.	138	137	143	474	480	482	3′495	3′242	3′449
	Med.	135	135	140	465	476	481	3′568	3′236	3′420
TGen ($\mathcal{U}$)	Avg.	5′349	26′338	53′578	8′879	44′162	88′709	19′784	89′023	177′998
	Med.	5′319	26′210	53′392	8′802	43′437	88′400	20′279	87′714	176′417
TGen ($\mathcal{T}$)	Avg.	1′197	3′450	6′396	2′616	6′342	11′124	8′894	15′677	25′054
	Med.	1′187	3′423	6′342	2′592	6′272	11′063	9′068	15′445	24′851
Comm ($\mathcal{U}$)	Avg.	786	786	799	1′204	1′215	1′226	2′505	2′294	2′305
	Med.	782	785	794	1′200	1′206	1′221	2′580	2′264	2′289
Comm ($\mathcal{S}$)	Avg.	1′667	1′667	1′702	2′716	2′744	2′767	5′957	5′431	5′466
	Med.	1′660	1′658	1′695	2′697	2′720	2′759	6′122	5′388	5′426

even "one-time") operations, while communication between a user and a service offers realistic performance in both protocols, even in the light that this implementation was not optimized in any way. The last statement becomes even more clear considering the case that five or ten ticket-granting servers are a rather pessimistic choice, while a bi-directional channel is then bootstrapped. In other words, with a more realistic setting with two or three ticket-granting servers, the run-time is entirely practical.

A more detailed description of the measurements is given in Appendix E.

3.6 Additional Extensions to the Protocols

The protocols given already offer the basic functionality of SSO, including a lot of privacy guarantees. However, from a practical perspective, one might think of some additional features, which one may want to have in a real deployment.

This section is devoted to discuss some of these potential additional requirements and how to alter the functionalities (and the corresponding protocols) to address them.

3.6.1 (t, n)-Threshold Version

In the current formalizations, all service providers want a proof that all ticket-granting servers approve that the user can access the service in question. This can be relaxed in a quite natural way. In particular, one only requires that, say, only $t < |\mathsf{T}|$ ticket-granting servers need to agree that a given user can access the service in question.

The ideal functionality needs to be adjusted as follows: instead of granting access to the services for which each ticket-granting servers agreed upon (for each *qid*), it stores the access rights for the services for which at least t servers grant the token. In protocol, the only change required is that not knowledge of all signatures is proven, but only t of them (with the same constraints), and for different (service providers') public keys. This can even be extended to hide which servers are used to prove that the access rights are actually given, e.g., by hiding the public keys used as well.

This can even be extended to a case where only t-out-of-n ticket-granting servers need to participate during the password check. Altering the protocol is easy: one does not generate the threshold encryption keys for the n-out-of-n case, but for the new t-out-of-n one. However, in this case already t corrupt servers learn the password pwd used at registration, while all other security guarantees still hold. Altering the functionality and protocols is straightforward. Note, however, that in the second protocol also the secret-sharing scheme SS has to be adjusted to the t-out-of-n case.

3.6.2 Adding Context-Information

The only information the ticket-granting servers can pass to the service providers is that a given uid has successfully obtained a token. This, however, may not always be sufficient. For example, it may be needed that the ticket-granting servers can pass some additional information, i.e., context, to the service providers.

This can easily be achieved by adding an additional parameter to the TOKENGENGRANT-interface, which is then also signed in the token-generation protocol. This context information is then simply made public, and attached to the sent message (or encrypted, if the context is to be hidden from the adversary/outside world).

3.6.3 Completely Unlinkable & Scope-Exclusive Tokens

Clearly, the services learn, by definition, the party id $\mathcal{U}$ of the party sending requests and the uid in the case of a corrupted service.

This may not always be wanted due to privacy concerns. However, implementing the change is quite straightforward: in the ideal functionality, this information is no longer given to the adversary. For the protocol, however, one has to resort to some form of onion-routing to hide the origin of the messages, as done for anonymous credentials [CLNR14, CL05]. However, in this case, one has to take extra care that the *qid*'s are also generated in an unlinkable way.

Moreover, a corrupt user can generate as many pseudonyms for a given scope as it wants to. However, as scope-exclusive pseudonyms are deterministic, it seems to be very hard to correctly simulate the secret key usk once it needs to be given to the adversary, especially if the simulator does not know which token was used and thus needs to somehow "program" usk which cannot be exponentially big. Also see Appendix C and Appendix D, where two scope-exclusive pseudonym-systems are presented.

3.6.4 Adding New Services

The protocol enforces that the services are fixed at setup. However, a simple alteration allows to add additional services (note, the access rights are input by the environment anyway). In particular, one removes S from the *sid*, and solely bases access to the services on the environment's input. Note, however, that the message signed needs to contain which service provider is meant – in the proposed protocols, this is encoded over the position in the message, which is then no longer possible. A solution is simple: one signs $\mathcal{S}$ instead of 1 in the first protocol, while signing $\mathcal{S}$ mapped to group element using a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_1$ in the second protocol anyway. Note, however, that in this case the size of the encryption may leak how many service providers a user can access in the second protocol — this can be circumvented if there is an upper bound on the services a given user can access with each token, while the encrypting party signs dummy values.

Clearly, all these extensions can be combined, while the proofs of security carry over with only minor adjustments. However, it was chosen not to incorporate the changes directly to make the protocols, proofs and definitions more readable.

3.7 Conclusion

This chapter introduced the notion of password-based distributed UC-secure single sign-on, where token generation and password-checking is split up among multiple ticket-granting servers to protect the users' passwords. Two ideal functionalities $\mathcal{F}_{\mathsf{sso}}^1$ and $\mathcal{F}_{\mathsf{sso}}^2$, along with protocols realizing them, were given. In both, the adversary does not learn anything about the (wrong) password-attempts, and all ticket-granting servers need to be corrupt before the adversary learns the password used at registration. The ideal functionality $\mathcal{F}_{\mathsf{sso}}^1$ offers the basic functionality one expects, while $\mathcal{F}_{\mathsf{sso}}^2$ offers additional privacy guarantees such as unlinkable tokens and incorporates pseudonyms, at the cost being slightly less efficient.

However, a still open problem is to find a suitable and provably secure way how to update passwords and to make the tokens scope-exclusive.

Chapter 4

Virtual Smart-Cards: Signing With a Password and a Server

The results of this chapter have already been published [CLNS16].

Abstract. An important shortcoming of client-side cryptography on consumer devices is the poor protection of secret keys. Encrypting the keys under a human-memorizable password hardly offers any protection when the device is stolen. Trusted hardware tokens such as smart-cards can provide strong protection of keys but are cumbersome to use. This chapter considers the case where secret keys are used for digital signatures and proposes a password-authenticated server-aided signature Pass2Sign protocol, where signatures are collaboratively generated by a device and a server, while the user authenticates to the server with a (low-entropy) password. Neither the server nor the device store enough information to create a signature by itself or to perform an offline attack on the password. The signed message remains hidden from the server. It is argued that the protocol offers comparable security to trusted hardware, but without its inconveniences. It is also proven secure in the universal composability (UC) framework in a new corruption model where, unlike standard UC, the adversary does not obtain past inputs and outputs upon corrupting a party. This is crucial to hide previously entered passwords and messages from the adversary when the device gets corrupted. The protocol itself is surprisingly simple: it is round-optimal, efficient, and relies exclusively on standard primitives. The security proof involves a novel random-oracle programming technique.

Roadmap. Section 4.1 contains the problem statement, while additional preliminaries are presented in Section 4.1.3. The ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$ is presented in Section 4.2. The corresponding protocol is given in Section 4.3, while the protocol is proven secure in Section 4.4. The evaluation of the implementation is contained in Section 4.5, while a few modifications of the basic scheme are given in Section 4.6. This chapter is concluded in Section 4.7.

4.1 Introduction

Mobile devices such as smart-phones and tablets are used more and more for security-critical tasks such as e-banking, authentication, and signing documents. However, they can be infected

by malware and, due to their mobility, the devices are easily lost or stolen. Keeping cryptographic keys safe in such an environment is challenging. Typically, they are simply encrypted with a human-memorizable password. If a device is lost, stolen, or compromised by malware, the password-encrypted keys are usually easily recovered through an offline dictionary attack. Such attacks are extremely effective on modern hardware, especially given the low entropy in human-memorizable passwords [Gos12].

Higher-security use cases such as online banking or government-issued electronic identification (eID) therefore often resort to tamper-proof hardware such as smart-cards, SIM cards or trusted platform modules (TPMs) for extra protection. The hardware tokens offer interfaces to interact with the keys, e.g., to compute digital signatures on messages provided by the host, while the signing key never leaves the confined environment. Usually, a password or PIN code is added as a second layer of protection. Guessing attacks on the password or PIN are infeasible as the token blocks after too many failed attempts. In case a hardware token is compromised, it can additionally be rendered useless by revoking its public key.

The protection is not perfect though. Without a dedicated display, malware on the host machine may instruct the plugged-in token to sign more or different messages than the user intended to sign. Also, side-channel attacks such as differential power analysis only become more powerful with time. In other words, what is considered tamper-proof hardware today, may not be so anymore tomorrow [KK99]. Additionally, trusted hardware suffers from poor usability and high deployment and maintenance costs. Users find it inconvenient to carry a hardware token for each security-sensitive application. Desktop and laptop computers rarely come with built-in smart-card readers and not all consumer-grade machines have TPMs. External USB card readers are available, but supporting drivers and browser plug-ins on several platforms simultaneously requires a considerable effort. Using trusted hardware in combination with mobile devices is even more problematic, as they often lack connectivity to interact with external tokens.

So the question is, is there a way to realize similar security guarantees as hardware tokens yet avoiding their practical inconveniences? Software obfuscation [BGI+01, BGI+12, BM14, GGH+13] may come to mind, but does not help at all: leaking an obfuscated signing algorithm to an adversary is just as bad as leaking the signing key itself. As network connectivity is far more ubiquitous than trusted hardware in consumer devices, how about relying on the assistance of an online server to create signatures? A solution must protect the keys as long as at least one of the device or the server is not corrupted. Moreover, it is required that the user is only required to remember at most a potentially weak password or a PIN code, while offering protection against offline password guessing attacks. Involving an online server in the signing process enables additional control of the use of the signing key, as the server can block the account or involve a second authentication factor. However, it is stressed that this approach therefore requires network access and an always reachable server, which now also needs to be maintained. Thus, it depends on the use-case which approach one prefers.

4.1.1 Contribution

This chapter introduces *password-authenticated server-aided signatures* (Pass2Sign), where the signing key is distributed over the user's device and an online server. The signing key is never reconstructed; rather, the device and the server must engage in a distributed protocol to

compute signatures. For added security, the user must enter a password on the device each time a signature is generated. The server not only verifies the password, but also the identity of the device, i.e., an adversary without access to the device cannot even perform an online guessing attack. This prevents an adversary from blocking an honest user's account by swamping the server with fake login attempts: the server simply ignores signing attempts from the wrong device. If the device falls into the wrong hands, or if the device is compromised by malware, then the attacker must still perform an *online* guessing attack before it can generate signatures. When the server detects too many failed password attempts or signing requests per time period, it can take appropriate action such as blocking the account or requiring additional authentication. The server neither learns the message that is being signed, nor does it learn the user's password (or password attempts). This not only protects the user against malicious servers, but also protects the password in case the server is broken into by hackers.

Malware running on the device can of course capture both the device keys and the password, enabling the adversary to sign any messages it wants, but only by interacting with the server for each new signature. The server can therefore implement additional security measures on top of the protocol, e.g., a logic which detects abnormal signing behavior, or a secondary communication channel via which the server informs the user about his account activity. When suspicious transactions are detected, the server can block the user's account to ensure that no further signatures can be created, and revoke the user's public key.

The resulting security level is almost identical to the protection offered by trusted hardware tokens, but without their inconveniences. Only few smart-card readers feature integrated trusted keypads and displays; built-in secure elements such as SIM cards or TPMs never do. Malware running on the host system can therefore also capture the user's PIN code and have different messages signed than what is shown on the screen. The main security guarantee of trusted hardware tokens is therefore that no more signatures can be generated after unplugging the token—which can actually be quite cumbersome or even impossible for SIM cards and TPMs. In the same way, the protocol prevents further signatures from being generated when the user's account is blocked by the server. When the device is lost, the given solution even offers better protection than hardware: while it may be possible to extract the keys from a compromised token, the only information that an adversary can extract from a corrupted device is a "useless" key share.

Strong Security Notion and Corruption Model. The security of the Pass2Sign scheme is defined in the universal composability (UC) framework [Can01]. The main goal of the protocol is to guarantee protection of the user's password and signing key in the event of device or server compromise. Therefore, a very strong corruption model is proposed that, unlike standard corruptions as defined in the UC framework, does not hand all past inputs and outputs to the adversary when a party is corrupted. In case the device gets corrupted, these inputs include the user's password and all previously signed messages, which obviously goes directly against the security goals. Clearly, it is impossible to achieve such a strong corruption model without secure erasures: if the entered passwords are not erased, then there is no way to hide it from an adversary upon corruption.

As already discussed in Chapter 1, the UC-framework is well-known to provide superior and more natural security guarantees for the particular case of password-based protocols than traditional game-based notions [CHK+05b]. In particular, by letting the environment generate

all passwords and password attempts, UC formulations correctly model arbitrary dependencies between passwords. For example, their game-based counterparts fail to provide any security guarantees when honest users make typos while entering their passwords, a rather frequent occurrence in real life. Also, by absorbing password guessing attacks inside the functionality, secure composition with other protocols is guaranteed to hold; this is much less clear for game-based notions that tolerate a non-negligible adversarial success probability.

Efficient Protocols. One might expect that meeting such stringent security standards comes at a considerable cost in efficiency. Indeed, similar protocols involve a factor 4–10 in performance penalty to protect against (standard) adaptive corruptions [CEN15], while generic techniques to obtain adaptive security at least double the number of communication rounds [Ven14]. Blindness for signed messages is another feature that is notoriously expensive to achieve in the UC framework [KZ08]. It is therefore even more surprising that the protocol, in the random-oracle model, is refreshingly simple, round-optimal, and efficient. Generating a signature requires only three modular exponentiations on the device and two on the server, plus a few hash function evaluations, with only one protocol message from the device to the server and back. The resulting signature is an RSA-FDH signature on a double salted hash of the message and some technical values such as session identifiers. The protocol is also altered to two simpler variants for the setting where message blindness is not required. The first variant exposes the message only to the server, while the second variant does not hide the message at all.

Proof Technique. In spite of its simplicity, the security proof of the protocol is actually quite intricate. The many cases triggered by adaptive corruptions (which are allowed even during setup and arbitrarily interleaved signing sessions) and the mixture of passwords, encryption, and signatures require very careful bookkeeping, especially of the random-oracle responses during simulation. This proof also incorporates an interesting and novel technique to reconcile the seemingly contradictory requirements that the simulator must be able to determine the value of each signature, but without learning the message being signed. Typical blind signature techniques and the associated "one-more"-type security assumptions [BNPS03] are avoided by letting the device and the server both contribute to the randomness of the signature, and by programming the random oracle "just-in-time" at the moment that the signature is verified, not when it is created. This technique is of independent interest and may find applications in other scenarios as well.

Implementation. The protocol was implemented on a commodity mobile device to provide a thorough performance analysis of the prototypical implementation to demonstrate its practicality. With signature generation for a 2048-bit key requiring about 250ms in total, the protocol is clearly efficient enough for use in practice. The detailed results are given in Section 4.5.

4.1.2 Related Work

The idea of distributing signing keys over two separate entities dates back to Boyd [Boy89], and was later generalized to threshold signatures [DF89, Rab98, GRJK00, BDTW01, ADN06]. Threshold signatures assume that all parties agree on the messages signed, i.e., there is no "main entity" that can trigger a signing protocol and thus also no authentication to ensure

that only the "main entity" can do so. Bellare and Sandhu [BS01a] present two-party RSA signature protocols between a client and a server. However, in their protocols the server cannot be corrupted and requires no client-authentication.

Server-assisted signatures with additional password protection have been presented in the literature as well. Ganesan [Gan95] proposes a protocol where the client's signing exponent is derived from his password. This is similar to Gjøsteen and Thuen [GT11, Gjø13] who propose password-based signature schemes where the secret signing key is shared between different entities, but where one share only depends on the password. A server knowing the high-entropy part of the signing key can mount offline attacks by creating a signature based on a guessed password (from which the low-entropy key-share is derived) and verifying it under the public key. Hence, the above approaches assume that the server cannot be corrupted. The scheme offers much weaker security guarantees than the one introduced in the chapter: the server can mount offline attacks against the user's password and sign messages in the name of the user, as it knows the signing key. Both is not possible in the presented protocol. Xu and Sandhu [XS03] present two server-assisted threshold signature schemes for a closely related setting where a user can generate the signatures with the help of his device and a threshold of multiple servers. Their constructions do not yield single-server instantiations as a special case, however, because in their setting a collusion of servers larger than the threshold can perform an offline attack on the user's password, without access to the user's device.

Another line of work are the schemes given by Mannan and van Oorschot [MvO07], as well as the ideas presented by Damgård and Mikkelsen [DM09], which assume a trusted, yet resource-contrained device and aim at securely outsourcing parts of the device's computation to an untrusted entity. However, the goal is not to reduce the computational complexity for the trusted device, but to fully remove the assumption of trusted hardware.

The S-RSA protocol due to MacKenzie and Reiter [MR03] envisages many of the goals that are pursued here, such as requiring the adversary to compromise the device to perform even an online attack, avoiding offline attacks as long as the server and the device are not both compromised, and "key disabling" by blocking a user's account on the server. Their protocol is proven secure in a non-UC notion that is weaker than the one given in this chapter. Foremost, it does not enjoy the many advantages of UC password-based protocols discussed earlier, such as preserving security in case of mistyped passwords and secure composition with other protocols. Also, the server in their protocol sees the message being signed, can only be corrupted between (and not during) signing sessions, and can perform an offline dictionary attack on the password based on the information it sees during the signing protocol (but the last problem is easy to fix).

The protocol presented here is the first to support fully adaptive corruptions of the server as well as the device in the UC model. One could of course evaluate the signing algorithm using generic adaptively UC-secure multiparty computation (MPC), but this comes at great cost: evaluating even a *single* multiplication gate in the most efficient two-party computation protocol secure against adaptive corruptions [CES13] incurs computation and communication costs that are magnitudes larger than the given direct construction.

A more remotely related primitive is password-authenticated secret sharing (also called PASS-schemes), e.g., the schemes by Bagherzandi et al. [BJSL11] and Camenisch et al. [CEN15, CLLN14, CLN12]. A PASS-scheme allows a user to share a secret among a set of servers that it can later recover based on a password. In a sense, one can consider storing a secret signing key using a two-out-of-two PASS scheme, where the user's device plays the role of one of the

servers. This does not fulfill the given requirements, though, as the device would need to locally reconstruct the signing key in order to compute a signature. Moreover, once the signing key is reconstructed, the device can sign as many messages as it wants to without any additional checks.

4.1.3 Additional Building Blocks

For this chapter, some additional building blocks are required. These are presented next.

4.1.3.1 Receiver Non-Committing Labeled Public-Key Encryption

To deal with adaptive corruptions, a non-committing encryption scheme for the *receiver* is required. In the security proof, the simulator needs to be able to simulate ciphertexts without knowing the corresponding plaintexts which would be encrypted in the real protocol. However, when the adversary later corrupts the receiver of a simulated ciphertext, the simulator has to provide a state of the corrupted party such that all the ciphertexts decrypt to some concrete plaintext. This is related to the "selective de-commitment problem" [BH92].

The notion of non-committing encryption that is required is stronger than some that were proposed in the literature [FHKW10, HPW15] and weaker than others [Nie02]. To minimize the security assumptions for the protocol and open the possibility for more efficient instantiations, a new definition is introduced, which is a non-interactive construction in the random-oracle model.

A labeled non-committing encryption scheme ENC has the same interfaces as a standard encryption scheme. However, it has some additional, non-standard, security properties which are required for security of the protocol. Namely, the $\mathsf{RECV\text{-}SIM}$ security property is now introduced that a labeled non-committing encryption scheme needs to satisfy in this context.

Definition 4.1 (RECV-SIM security)**.** *An encryption scheme* ENC *is RECV-SIM-secure if for all PPT adversaries* $\mathcal{A}$ *there exists a stateful PPT simulator* $\mathsf{SIM_{NCE}}$ *and a negligible function* ν *such that:*

$$\left| \Pr[\mathsf{RECV\text{-}SIM\text{-}real}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{RECV\text{-}SIM\text{-}ideal}^{\mathsf{ENC}}_{\mathcal{A},\mathsf{SIM_{NCE}}}(\lambda) = 1] \right| \leq \nu(\lambda)$$

The corresponding experiments are depicted in Figures 4.1 and 4.2.

This definition says an encryption scheme is $\mathsf{RECV\text{-}SIM}$-secure (**REC**ei**V**er-**SIM**ulatable), if there exists a simulator that is given control over the random oracle such that no adversary can distinguish between simulated ciphertexts (which do not contain any information) and honestly generated ones.

More precisely, the adversary must not be able to tell in which of the two experiments it is run in, even if it can adaptively query for new encryptions, decryptions, and receives the secret key at some point. The crucial difference is that in the ideal game the adversary receives simulated ciphertexts instead of real ones, which are computed by a simulator on input only the length of the plaintext. Moreover, the adversary $\mathcal{A}$ receives the secret key of the receiver at a later point of its own choice. In the ideal world, this secret key is provided by the simulator which only now learns the plaintexts to which it produced the simulated ciphertexts. The

Experiment $\mathsf{RECV\text{-}SIM\text{-}ideal}^{\mathsf{ENC}}_{\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}}}(\lambda)$:
$\mathsf{pk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{PUBLICKEY}, 1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$state_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot),\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}}(\cdot,\cdot),\mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}}(\cdot,\cdot)}(\mathsf{pk}_{\mathsf{ENC}})$
where oracle $\mathsf{Enc}_{\mathsf{ENC}}$ on input m_i and ℓ_i:
$c_i \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPT}, |m_i|, \ell_i)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c_i, m_i, \ell_i)\}$
return c_i
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input c_j and ℓ_j:
if $(c_j, m_j, \ell_j) \in \mathcal{Q}$, return m_j
return $m_j \leftarrow \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{DECRYPT}, c_j, \ell_j)$
where oracle $\mathcal{H}$ on input q_k:
return $h_k \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, q_k)$
$\mathsf{sk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$
return $\mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot)}(\mathsf{sk}_{\mathsf{ENC}}, state_{\mathcal{A}})$

Figure 4.1: Experiment RECV-SIM-ideal for the RECV-SIM definition

adversary expects that the ciphertexts indeed decrypt to the messages queried to the encryption oracle using the given secret key.

Compared to the definition given by Fehr et al. [FHKW10], the adversary $\mathcal{A}$ is allowed adaptive queries, and receives the secret key $\mathsf{sk}_{\mathsf{ENC}}$ (but not the randomness used to create it) at the last step of the experiment, while the definitions given by Hazay et al. [HPW15] only consider a single (or randomly sampled according to some distribution) message per key pair, which is not enough for the protocol. Likewise, Nielsen [Nie02] gives a formulation of non-committing encryption—in the sense of secure message transmission—in the UC framework. However, his functionality is stronger than what is needed here, because it simulates all randomness, which is not required, as the protocol relies on secure erasures anyway.

Note, in Chapter 5 is shown how to achieve an even stronger notion which does not require secure erasures, and *no* interaction, i.e., does not model secure message transfer as Nielsen does [Nie02].

Analogously to Fehr et al. [FHKW10], it is now shown that this definition implies standard IND-CCA2 security.

Theorem 4.2. *Any encryption scheme that is RECV-SIM-secure is also IND-CCA2-secure.*

Proof. Intuitively, the proof formalizes the following idea: if an adversary can decide which message a given ciphertext contains, then it can decide whether a given ciphertext contains any information at all.

Assume an (efficient) adversary $\mathcal{A}$ that guesses the bit b in the IND-CCA2 game with probability at least $\frac{1}{2} + \epsilon$. Then there is a construction of an (efficient) adversary $\mathcal{B}$ such that the probability that it outputs 1 in the two experiments RECV-SIM-real and RECV-SIM-ideal differs at least by ϵ. The proof is straightforward. Essentially, the reduction feeds the ciphertexts to

Experiment $\mathsf{RECV\text{-}SIM\text{-}real}^{\mathsf{ENC}}_{\mathcal{A}}(\lambda)$:
$(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$
$state_{\mathcal{A}} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot),\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}}(\cdot,\cdot),\mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}}(\cdot,\cdot)}(\mathsf{pk}_{\mathsf{ENC}})$
where oracle $\mathsf{Enc}_{\mathsf{ENC}}$ on input m_i and ℓ_i):
return $c_i \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m_i, \ell_i)$
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input c_j and ℓ_j):
return $m_j \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c_j, \ell_j)$
where oracle $\mathcal{H}$ on input q_k:
return $h_k \leftarrow \mathcal{H}(q_k)$
return $\mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot)}(\mathsf{sk}_{\mathsf{ENC}}, state_{\mathcal{A}})$

Figure 4.2: Experiment RECV-SIM-real for the RECV-SIM definition

$\mathcal{A}$, and if $\mathcal{A}$ guesses correctly, then it must be the real experiments as the ciphertexts in the ideal experiments are unrelated to the challenge message. More precisely, the reduction is as follows. Initially, $\mathcal{B}$ receives a public key $\mathsf{pk}_{\mathsf{ENC}}$ of the encryption scheme, oracle access to $\mathsf{Enc}_{\mathsf{ENC}}$, $\mathsf{Dec}_{\mathsf{ENC}}$, and a random oracle $\mathcal{H}$. Thus, $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{pk}_{\mathsf{ENC}}$. Whenever $\mathcal{A}$ requests access to its decryption oracle, $\mathcal{B}$ forwards $\mathcal{A}$'s query to its own oracle, and returns the unmodified answer to $\mathcal{A}$. When eventually $\mathcal{A}$ outputs the challenge messages $((m_0, m_1), \ell, state_{\mathcal{A}})$, $\mathcal{B}$ checks that $|m_0| = |m_1|$, and that both $m_0 \in \mathcal{MS}$ and $m_1 \in \mathcal{MS}$. If any checks fail, $\mathcal{B}$ sets $c \leftarrow \perp$. Else, $\mathcal{B}$ picks a random bit $b \in \{0,1\}$ and calls its own encryption oracle: $c \leftarrow \mathsf{Enc}_{\mathsf{ENC}}(m_b, \ell)$. $\mathcal{B}$ then passes $(state_{\mathcal{A}}, c)$ to $\mathcal{A}$. If now (c, ℓ) is queried to the decryption oracle, $\mathcal{B}$ responds with $\perp$. For every other query, $\mathcal{B}$ uses its own decryption oracle, and forwards the answer to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs its guess b^*. $\mathcal{B}$ outputs 1 if $b^* = b$, and 0 otherwise. Let us analyze the probability that $\mathcal{B}$ outputs 1 in both experiments.

1. If $\mathcal{B}$ is run in the experiment RECV-SIM-real, then clearly, $\mathcal{B}$ was playing the IND-CCA2 game with $\mathcal{A}$, using the random bit b. Hence, it holds that $\Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM\text{-}real}}_{\mathsf{ENC},\mathcal{B}} = 1] = \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{ENC},\mathcal{A}} = b'] = \frac{1}{2} + \epsilon$.
2. If $\mathcal{B}$ run in experiments RECV-SIM-ideal, then the challenge ciphertext that $\mathcal{B}$ fed to $\mathcal{A}$ was unrelated to m_b, i.e., the answer b^* of $\mathcal{A}$ is information-theoretically independent of b and therefore $\Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM\text{-}ideal}}_{\mathsf{ENC},\mathcal{B}} = 1] = \frac{1}{2}$ follows.

So, the difference between these two probabilities is at least ϵ. Note, the definition of RECV-SIM also has access to a random oracle. This is the reason why the random oracle was made explicit in the IND-CCA2 definition. □

Instantiation. Now, a concrete instantiation for an encryption scheme that achieves RECV-SIM security is given. Namely, the IND-CCA2-secure encryption scheme introduced by Bellare and Rogaway [BR93] is modified to include labels and handle arbitrary-length messages. Let $\mathcal{G} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathcal{K} : \{0,1\}^* \to \{0,1\}^\lambda$ denote two hash functions, modeled as random oracles. Further, an encoding scheme $\mathsf{EC} = (\mathsf{ec}, \mathsf{dc})$ which allows to map arbitrary length messages to a list of blocks with fixed length is required. More precisely, let $\mathsf{ec} : \{0,1\}^* \to (\{0,1\}^\lambda)^+$ be a

injective encoding function and $\mathsf{dc} : (\{0,1\}^\lambda)^+ \to \{0,1\}^*$ be the corresponding decoding function that returns $\perp$ if no valid pre-image exists. It is required that both functions are computable in polynomial time and that the output length of ec only depends on the length of its input, while dc and ec need to be perfectly correct, i.e., for all $\lambda \in \mathbb{N}$, and all messages $m \in \{0,1\}^*$ it holds that $m = \mathsf{dc}(\mathsf{ec}(m))$ with probability one. Note, it is not required that these functions are deterministic.

Construction 4.3 (RECV-SIM-secure ENC)**.** *Let* ENC *be constructed as follows:*

$\mathsf{KeyGen}_{\mathsf{ENC}}$**.** *Generate the key pair in the following way:*

1. *Generate a random* TDP *, i.e.,* $(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$*. The message space* $\mathcal{MS}$ *is* $\{0,1\}^*$*.*
2. *Output the public key* $\mathsf{pk}_{\mathsf{ENC}} = (f, \Sigma)$*, and* $\mathsf{sk}_{\mathsf{ENC}} = f^{-1}$ *as the secret key.*

$\mathsf{Enc}_{\mathsf{ENC}}$**.** *To encrypt a message* m *w.r.t. to* $\mathsf{pk}_{\mathsf{ENC}}$ *and label* ℓ *do:*

1. *Let* $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{ec}(m)$*.*
2. *Draw* $x \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(\Sigma)$*, compute* $c_1 \leftarrow f(x)$*,* $c_2^i \leftarrow \mathcal{G}(i, x) \oplus m_i$ *for* $i = 1, 2, \ldots, k$*, and* $c_3 \leftarrow \mathcal{K}(x, k, m, \ell)$*.*
3. *Output the ciphertext* $c \leftarrow (c_{(1)}, (c_{(2)}^1, c_{(2)}^2 \ldots, c_{(2)}^k), c_{(3)})$*.*

$\mathsf{Dec}_{\mathsf{ENC}}$**.** *To decrypt a ciphertext* c *w.r.t. to* $\mathsf{sk}_{\mathsf{ENC}}$ *and label* ℓ *do:*

1. *Parse* c *as* $(c_{(1)}, (c_{(2)}^1, c_{(2)}^2, \ldots, c_{(2)}^{k'}), c_{(3)})$ *for some* $k' \geq 1$*.*
2. *Compute* $x' \leftarrow f^{-1}(c_{(1)})$ *and* $m_i' \leftarrow \mathcal{G}(i, x') \oplus c_{(2)}^i$ *for* $i = 1, 2, \ldots, k'$*.*
3. *Let* $m' \leftarrow \mathsf{dc}(m_1', \ldots, m'_{k'})$*.*
4. *If* $m' = \perp$ *or* $c_{(3)} \neq \mathcal{K}(x', k', m', \ell)$*, output* $\perp$*.*
5. *Output* m'*.*

The above construction is clearly perfectly correct, while the proof of the following theorem is given in Appendix F.

Theorem 4.4. *The construction above is RECV-SIM-secure, if* $\mathcal{G}$ *and* $\mathcal{K}$ *are modeled as random oracles and* $\mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$ *is a secure* TDP *generator.*

It is shown in Chapter 5 that a slightly altered version of the construction above achieves much stronger security, while also having a tighter reduction.

4.2 Ideal Functionality

Now the password-authenticated server-aided signatures (Pass2Sign) are formally defined by describing its ideal functionality in the universal composability (UC) framework [Can01].

First, recall the high-level goal of the Pass2Sign scheme: signatures on messages are derived collaboratively between two parties, a device $\mathcal{D}$ and a server $\mathcal{S}$, meaning that a valid signature can only be obtained if both parties agree to the generation. Access to the server's signing operation is protected by a user password pwd that is chosen at setup and needs to be provided

for every signing request. The server then verifies whether the password is correct and also whether the request came from the correct device, which has the additional advantage that an attacker cannot block an honest user's account by swamping the server with false login attempts. The protocol must be secure against offline attacks on the password used during setup and on the password attempts during signing. That is, as long as at least one party remains honest, the adversary does not learn anything about the used passwords. In particular, the server learns only whether a password attempt in a signing request was correct or not, but not the actual password attempt itself. The server also does not learn the messages being signed. If this blindness feature is not required, Section 4.6 discusses how it can easily be removed. Security must be guaranteed for adaptive corruptions in order to protect against the main threat, namely the user losing his device. Note that the user of the device is subsumed into the environment to have a more readable functionality. How this maps to real-life scenarios is discussed at the end of this section.

1. `Init Server.` On input (INIT, sid) from server $\mathcal{S}$:
   - Create record $(\mathsf{initeq\text{-}rec}, sid)$.
   - Send (INIT, sid) to $\mathcal{A}$.
2. `Setup Request Device.` On input $(\mathsf{SETUPREQ}, sid, \mathsf{pwd})$ from device $\mathcal{D}$:
   - If there is a record $(\mathsf{setupreq\text{-}rec}, sid, \mathsf{pwd})$, ignore.
   - Create a record $(\mathsf{setupreq\text{-}rec}, sid, \mathsf{pwd})$.
   - Send $(\mathsf{SETUPREQ}, sid)$ to $\mathcal{A}$.
3. `Key Generation.` On input $(\mathsf{KEYGEN}, sid, \mathsf{pwd}^*, \mathsf{pk})$ from adversary $\mathcal{A}$:
   - If there is no record $(\mathsf{setupreq\text{-}rec}, sid, \mathsf{pwd})$, ignore.
   - If there is no record $(\mathsf{initeq\text{-}rec}, sid)$. ignore.
   - If $\mathcal{D}$ (taken from sid) is corrupt, then mark this instance as $\mathsf{key\text{-}corrupt}$.
   - If $\mathcal{D}$ is corrupt and $\mathsf{pwd}^* \neq \perp$, then create a record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}^*, \mathsf{pk})$. Else, create a record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}, \mathsf{pk})$.
   - Output $(\mathsf{SETUP}, sid, \mathsf{pk})$ to $\mathcal{D}$.

Figure 4.3: Setup interfaces of the ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$

The detailed description of the ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$ for password-authenticated server-aided signatures is given in Figure 4.3, Figure 4.4, and Figure 4.5. When describing the functionality, the following conventions to reduce repetitive notation is used:

- For the INIT, SETUPREQ and KEYGEN interfaces, the ideal functionality only considers the first input for each sid. Subsequent inputs to the same interface for the same sid are ignored. For the SIGNREQ, DELIVER, PROCEED, SIGNATURE interfaces the functionality only considers the first input for each combination of sid and qid.
- At each invocation, the functionality checks that $sid = (\mathcal{S}, \mathcal{D}, sid')$ for some server identity $\mathcal{S}$, device identifier $\mathcal{D}$ and $sid' \in \{0,1\}^*$. Also, whenever $\mathcal{F}$ receives input from or provides output to $\mathcal{S}$ or $\mathcal{D}$, $\mathcal{S}$ or $\mathcal{D}$ as specified in the sid, respectively, are meant.
- If the functionality "looks up a record", it is implicitly understood that if the record is not found, the functionality ignores the input and returns control to the environment.
- The session (sid) and query identifiers (qid) given as input to the functionality must be globally unique. In the two-party setting that is considered, this can be achieved by exchanging random

4. `Sign Request.` On input $(\mathsf{SIGNREQ}, sid, qid, \mathsf{pwd}', m)$ from device $\mathcal{D}$:
   - If there is no record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}, \mathsf{pk})$, ignore.
   - Create a record $(\mathsf{signreq\text{-}rec}, sid, qid, \mathsf{pwd}', m)$.
   - Send $(\mathsf{SIGNREQ}, sid, qid)$ to $\mathcal{A}$.
5. `Sign Delivery.` On input $(\mathsf{DELIVER}, sid, qid, \mathsf{pwd}^*, m^*)$ from adversary $\mathcal{A}$:
   - If there is no record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}, \mathsf{pk})$, ignore.
   - If there is no record $(\mathsf{signreq\text{-}rec}, sid, qid, \mathsf{pwd}', m)$, ignore.
   - If $\mathcal{D}$ is corrupt and $\mathsf{pwd}^* \neq \bot$, then set $\mathsf{pwd}' \leftarrow \mathsf{pwd}^*$ and $m \leftarrow m^*$.
   - If $\mathsf{pwd}' = \mathsf{pwd}$ then set $\mathsf{status} \leftarrow \mathsf{pwdok}$, else set $\mathsf{status} \leftarrow \mathsf{pwdwrong}$.
   - Create a record $(\mathsf{sign\text{-}rec}, sid, qid, m, \mathsf{status})$.
   - Output $(\mathsf{SIGNREQ}, sid, qid, \mathsf{status})$ to $\mathcal{S}$.
6. `Server Proceed.` On input $(\mathsf{PROCEED}, sid, qid)$ from server $\mathcal{S}$:
   - If there is no record $(\mathsf{sign\text{-}rec}, sid, qid, m, \mathsf{status})$ with $\mathsf{status} = \mathsf{pwdok}$, ignore.
   - Update the record to $\mathsf{status} \leftarrow \mathsf{PROCEED}$, and send $(\mathsf{PROCEED}, sid, qid)$ to $\mathcal{A}$.
7. `Signature Generation.` On input $(\mathsf{SIGNATURE}, sid, qid, \sigma)$ from $\mathcal{A}$:
   - If there is no record $(\mathsf{sign\text{-}rec}, sid, qid, m, \mathsf{status})$, ignore.
   - If $\mathcal{S}$ is honest, only proceed if $\mathsf{status} = \mathsf{PROCEED}$.
   - If there is no record $(\mathsf{signature\text{-}rec}, \mathsf{pk}, m, \sigma, \mathsf{false})$, then create a record $(\mathsf{signature\text{-}rec}, \mathsf{pk}, m, \sigma, \mathsf{true})$, and output $(\mathsf{SIGNATURE}, sid, qid, \sigma)$ to $\mathcal{D}$.
8. `Verify.` On input $(\mathsf{VERIFY}, sid, \mathsf{pk}', m, \sigma)$ from a party $\mathcal{P}$ or adversary $\mathcal{A}$:
   - Create a record $(\mathsf{verf\text{-}rec}, sid, \mathsf{pk}', m, \sigma, \mathcal{P})$ and send $(\mathsf{VERIFY}, sid, \mathsf{pk}', m, \sigma, \mathcal{P})$ to $\mathcal{A}$.
9. `Verified.` On input $(\mathsf{VERIFIED}, sid, \mathsf{pk}', m, \sigma, \phi)$ from $\mathcal{A}$ with $\phi \in \{\mathsf{true}, \mathsf{false}\}$:
   - If there is no record $(\mathsf{verf\text{-}rec}, sid, \mathsf{pk}', m, \sigma, \mathcal{P})$, ignore.
   - Delete record $(\mathsf{verf\text{-}rec}, sid, \mathsf{pk}', m, \sigma, \mathcal{P})$.
   - Record $(\mathsf{signature\text{-}rec}, \mathsf{pk}', m, \sigma, f)$ and output $(\mathsf{VERIFIED}, sid, \mathsf{pk}', m, \sigma, f)$ to $\mathcal{P}$, where f is determined as follows:
     - If a record $(\mathsf{signature\text{-}rec}, \mathsf{pk}', m, \sigma, f')$ for some f' exists, set $f \leftarrow f'$. *(consistency)*
     - Else, if a record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}, \mathsf{pk})$ exists with $\mathsf{pk} = \mathsf{pk}'$ and the instance is not marked $\mathsf{key\text{-}corrupt}$, set $f \leftarrow \mathsf{false}$. *(strong unforgeability)*
     - Else, set $f \leftarrow \phi$.

Figure 4.4: Main interfaces of the ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$

nonces between both parties and including the concatenation of both in the identifiers. Also, honest parties drop any inputs with session or query identifiers to which they did not contribute.

- An instance is "marked" a specified label is associated with the instance of the functionality with the current *sid*. This does not affect other instances of the functionality with a different *sid*.

Now, the behavior of all interfaces is described in a somewhat informal manner to clarify the security properties that the functionality provides.

Setup. The setup-related interfaces allow the device $\mathcal{D}$ to create an account that is protected with a password pwd and associated with the server $\mathcal{S}$.

1. The INIT interface allows the server $\mathcal{S}$ to generate and register any key pairs it requires for further interaction with the device $\mathcal{D}$.

10. `Corruption.` On input $(\mathsf{CORRUPT}, sid, \mathcal{P}, \Sigma)$ from adversary $\mathcal{A}$:
    - If there is no record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd}, \mathsf{pk})$, ignore.
    - Initialize a list $\mathcal{L} \leftarrow \emptyset$.
    - If $\mathcal{P} = \mathcal{S}$, then assemble $\mathcal{L}$ containing (qid_i, c_i) for all existing records $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}'_i, m_i)$, where $c_i \leftarrow \mathsf{pwdok}$ if $\mathsf{pwd} = \mathsf{pwd}'_i$ and $c_i \leftarrow \mathsf{pwdwrong}$ otherwise.
    - If now both $\mathcal{D}$ and $\mathcal{S}$ are corrupt, then mark this instance as $\mathsf{key\text{-}corrupt}$ and complete the abandoned sign requests: For all $(qid_i, \sigma_i) \in \Sigma$, look up m_i from record $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}'_i, m_i)$. If there does not exist a record $(\mathsf{signature\text{-}rec}, \mathsf{pk}, m_i, \sigma_i, \mathsf{false})$, then create a record $(\mathsf{signature\text{-}rec}, \mathsf{pk}, m_i, \sigma_i, \mathsf{true})$.
    - Send $(\mathsf{CORRUPT}, sid, \mathcal{P}, \mathcal{L})$ to $\mathcal{A}$.
11. `Password Guessing.` On input $(\mathsf{PWDGUESS}, sid, qid, \mathsf{pwd}^*)$ from adversary $\mathcal{A}$:
    - If not both $\mathcal{D}$ and $\mathcal{S}$ are corrupt, then ignore this input.
    - If $qid = \bot$ then look up a record $(\mathsf{setupreq\text{-}rec}, sid, \mathsf{pwd})$.
    - If $qid \neq \bot$ then look up a record $(\mathsf{signreq\text{-}rec}, sid, qid, \mathsf{pwd}, m)$.
    - Set $c \leftarrow \mathsf{pwdok}$, if $\mathsf{pwd}^* = \mathsf{pwd}$ and $c \leftarrow \mathsf{pwdwrong}$ otherwise.
    - Send $(\mathsf{PWDGUESS}, sid, qid, c)$ to $\mathcal{A}$.

Figure 4.5: Corruption interfaces of ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$

2. The SETUPREQ interface allows the device $\mathcal{D}$ to register with the server $\mathcal{S}$, where $\mathcal{D}$ and $\mathcal{S}$ are the identities as included in the session identifier.
3. The KEYGEN interface allows the adversary to complete the setup by determining the public key pk under which messages in this instance are signed. If the device $\mathcal{D}$ is corrupt at the time of key generation, then the instance is said to be key-corrupt, meaning that the adversary may know the signing key and may therefore be able to sign any messages it wants. A corrupt device $\mathcal{D}$ additionally has the option to "overwrite" the original password pwd from the SETUPREQ input (that may have been provided when $\mathcal{D}$ was still honest) with a new password pwd^*. This does not affect the unforgeability of signatures, as the instance is key-corrupt anyway, but does allow the adversary to later perform signing requests with correct passwords with $\mathcal{S}$ without having to guess pwd.

Signature Generation. Once setup is completed, the signature-related interfaces allow the device $\mathcal{D}$ to obtain a signature σ on a message m, but only if the provided password attempt pwd' is correct and the server $\mathcal{S}$ agrees to the signature generation. Signing requests can be done in parallel; the unique query identifier qid identifies different signing sessions.

4. The SIGNREQ interface allows the device $\mathcal{D}$ to submit the message m to be signed, and a password attempt pwd'. Only the device $\mathcal{D}$ included in sid can perform signing requests, so without compromising the device, the adversary cannot even perform an online attack against the setup password pwd.
5. The DELIVER interface lets the adversary notify the server of an incoming signing request. The notification only includes whether the submitted password is correct, but not the password attempt itself. A corrupt device $\mathcal{D}$ can overwrite the original password pwd' and message m from the SIGNREQ input (that may have been provided when $\mathcal{D}$ was still honest) with a new password pwd^* and message m^*.

6. The PROCEED interface allows the server to indicate whether it wants to proceed with the signing request. This models the opportunity for an external throttling mechanism to refuse the signing request. An honest server can only proceed if the password was correct, but corrupt servers can proceed regardless of whether the passwords matched. If the server is honest, then the adversary only (implicitly) learns whether the password was correct when the server agrees to proceed.
7. The SIGNATURE interface allows the adversary to determine the value σ of the signature and have it delivered to the requesting device. If the server is honest, then a signature can only be established if the server previously agreed to proceed. A corrupt server can choose to ignore an incorrect password. The functionality creates a signature record $(\mathsf{signature\text{-}rec}, \mathsf{pk}, m, \sigma, \mathsf{true})$ that will allow successful verification of the signature.

The above interfaces follow a similar approach to Canetti's signature functionality [Can04] where the adversary determines the signature value, with the important difference that the adversary does *not* learn the message being signed. This models a weak form of blindness: it ensures that a corrupt server does not learn the message, but it does not provide unlinkability as blind signatures do. A full blindness notion would let the functionality generate the signatures by running an algorithm provided by the adversary [Fis06]. Achieving such a notion is interesting, but would almost certainly come at a considerable overhead, as is the case for standard (UC-secure) blind signatures [Fis06, KZ08].

Signature Verification. With the verification interfaces, any party can check whether a signature σ is valid for message m and public key pk'.

8. The VERIFY interface allows any party $\mathcal{P}$ to ask for the verification of a signature σ on message m and under public key pk'.
9. The VERIFIED interface lets the adversary trigger the delivery of the verification result to $\mathcal{P}$ and also allows the adversary to input the verification result ϕ for adversarially controlled keys pk'. Here, adversarially controlled means that either pk' is different from the key pk registered in the functionality, or the instance for pk is key-corrupt. The ideal functionality enforces that responses are consistent, meaning that verification of the same signature for the same message and public key always returns the same result.
   For a non-adversarially-controlled key pk, the functionality guarantees strong unforgeability, meaning that even if the message m was signed before with signature σ, the adversary cannot come up with a different valid signature $\sigma' \neq \sigma$ for m. For regular unforgeability, one should add the condition that there does not exist a record $(\mathsf{signature\text{-}rec}, \mathsf{pk}', m, \sigma', \mathsf{true})$ for $\sigma' \neq \sigma$. It was opted for strong unforgeability because it offers more application scenarios and implies regular unforgeability [ADR02].

Corruptions. The functionality supports adaptive corruptions, i.e., the environment can, at any time, decide to corrupt any initially honest party. It is *not* a standard-corruption functionality as defined by the UC-framework [Can01] where the adversary, upon corruption of a party, obtains all the past inputs and outputs of that party. Such a corruption model is clearly unsuitable this setting, as it would hand the user password to the adversary as soon as the device gets corrupted. In fact, even when both the device and server are corrupted, it is

clearly insecure to give away the passwords immediately. Instead, the functionality then offers an interface modeling offline attacks against the passwords.

10. The CORRUPT interface allows the adversary to dynamically corrupt an initially honest device or server. When the device is corrupted after setup was complete, the adversary obtains the possibility to perform signing requests, but it still needs to provide the correct password. It is stressed that the adversary does not receive any passwords attempts or messages from past or even ongoing signing protocols.
    When the server gets corrupted, the adversary is told for all past signing requests whether or not the respective password attempts were correct. Still, the adversary is not given the stored password, nor the past password attempts themselves. Also, the adversary is not able to generate valid signatures.
    When both the device and the server are corrupt, the instance becomes key-corrupt, meaning that the adversary can sign any messages that it wants, and the adversary can offline-attack past passwords using the PWDGUESS interface described below. Additionally, the adversary can finish "abandoned"/hold-back sign requests, meaning sign requests that were never delivered to the server, that were overwritten by the adversary, that were turned down by the server because the password was incorrect, or for which the server did not (yet) agree to proceed. The adversary determines the signature values for abandoned requests in a separate input Σ. This interface may seem superfluous now that the instance is key-corrupt anyway, but the the adversary does not know the message of sign requests that were initiated when the device was still honest, so it cannot register these signatures through the normal VERIFIED interface. In the real world, an adversary who obtains the full state information of the device and server can inherently complete abandoned queries, so it has to be modeled here as well.
11. The PWDGUESS interface allows the adversary to perform offline attacks on the stored password and on previous password attempts, but only becomes available when both the device and the server are corrupt. For the stored password, offline attacks cannot be avoided, as the device and the server together must store some information that allows them to decide whether a password attempt was correct. For previous password attempts, this could in principle be avoided, but would make the protocol considerably less efficient, because new cryptographic material would have to be generated at each request and securely deleted afterwards.

If the device is already corrupt at the time of setup, the instance is considered as key-corrupt, even though the server might still be honest. A stronger security notion, requiring only slight changes to the functionality, would be achievable where the instance is only considered key-corrupt when both the device and server are corrupted. However, this would mean that in the realization, the key generation be done distributively between the server and the device. This is possible, but even for RSA rather inefficient [ACS02, HMRT12] and seems to offer little added security. Hence, it was chosen not to do this. However, while the realization of the set up would look quite differently, the functionality would not.

Discussion. Now is discussed how real-world attack scenarios map to the given ideal functionality. If a user loses his device, it is assumed that the adversary is able to extract *all*

non-erased the data from the device, so the device becomes corrupted. As long as the server is not corrupted, though, the adversary controlling the device still has to make online password guesses to be able to sign, but does not obtain the (full) signing key. To protect against online password guessing, the server should implement some kind of throttling on top of the protocol, such as refusing to serve further queries after too many failed password attempts.

If the device becomes infected by malware, the protocol also captures the worst case scenario: it may get *all* the (non-erased) data from the device and hence the device becomes corrupted. In contrast with the scenario above, the malware may also learn the (correct) password of the user if it is unaware of the infection and continues to use the device. This behavior is subsumed into the environment; this is modeled correctly by letting the environment provide the correct password to the adversary. Some protection against this kind of attack can be implemented on top of the protocol by adding intrusion detection logic on the server's side, e.g., by stopping to serve requests if they become too frequent. This situation is actually similar to that of a smart-card inserted in an infected device: the device could intercept the PIN and sign any messages it wants until the card is removed.

One could consider a more gradual corruption model where the device can be semi-corrupted, e.g., if an application turns malicious, but the uncompromised operating system separates it from other applications on the device. The model covers this as long as applications have their own protected execution space: the device in the model represents the application, while everything else is subsumed into the environment. More advanced models where applications can observe other applications (e.g., their running times) are beyond the scope of this chapter.

4.3 The Pass2Sign Protocol

The core idea of the protocol is fairly simple: an RSA secret key $d = d_{\mathcal{D}} \cdot d_{\mathcal{S}} \bmod \varphi(N)$ is split between the device and the server who then jointly perform the signing operation for each message m. To hide the message from the server, the device "blinds" it with randomness r as $h_m \leftarrow \mathcal{H}(r, m)$ and lets the server sign it as $\sigma_{\mathcal{S}} \leftarrow h_m^{d_{\mathcal{S}}}$. The device completes the signature as $\sigma \leftarrow \sigma_{\mathcal{S}}^{d_{\mathcal{D}}}$. For each signing request, the user authenticates towards the server using a salted password hash $h_p \leftarrow \mathcal{H}(k, \mathsf{pwd})$, where the salt k is stored on the device.

The Corruption Model and the Need for Secure Erasures. The main challenge is to maintain this simplicity while achieving the strong security properties envisaged. Most often, security against adaptive corruptions in the UC model comes at a considerable price in terms of computation and communication, while the presented corruption model is even substantially stronger. In particular, recall that the protocol must protect the user's password and previously signed messages in case the device is lost or stolen. The "standard corruption" model in the UC framework [Can01] (See Chapter 2) hands all previous inputs and outputs of a party to the adversary upon corruption of that party, which in case of the device would include all previous passwords and messages. It is quite obvious that standard corruption does not suffice here, and also that the model cannot be achieved without secure erasures, as there would be no way to securely erase previous inputs. Given the usual difficulty of achieving even standard UC corruption, it is surprising that the protocol remains refreshingly simple, round-optimal, and efficient.

Achieving Blindness. Achieving blind signatures against adaptive corruptions in the UC model is notoriously hard: the only scheme is due to Kiayias and Zhou [KZ08] and requires six rounds of communication and several zero-knowledge proofs. The presented protocol therefore strikes a reasonable compromise between security and efficiency by dropping the unlinkability requirement, i.e., the property that the signer cannot link a signature to a previous signing transcript, but focusing entirely on hiding the message from the signer. This also includes a new "just-in-time" programming technique for the random oracle that inserts the correct entries into the oracle when signatures are verified, rather than when they are created. This results in an efficient and round-optimal construction without having to rely on one-more-type assumptions that are typical for full-domain-hash blind signatures [BNPS03, Bol03].

In a bit more detail, to enable the simulator to open any signing transcript to any message-signature pair, the server adds another layer of randomness, i.e., it signs $h'_m \leftarrow \mathcal{H}(r', h_m)$ for some randomly chosen r'. When the simulator has to provide a signature σ to the functionality without knowing the message m, it simply signs a random value $h'_m \xleftarrow{\$} \{0,1\}^\tau$. The connection to m is only established when the signature is verified, which is coined "just-in-time" programming. Namely, whenever a random oracle query $\mathcal{H}(r', \mathcal{H}(r, m))$ is made where r, r' were previously used in a simulated blind signature, the simulator verifies whether (m, σ) is valid with the help of the ideal functionality. If so, the simulator programs the random oracle to map the message m it just learned to the randomly chosen h'_m that was signed as σ.

Non-Committing Communication and State. As UC allows corruptions *during* setup and signing sessions, one must take special care that messages sent by the device and server do not commit the simulator to values that it might not know at that time in the proof. This is achieved by employing non-committing encryption for the passwords hashes $h_p \leftarrow \mathcal{H}(k, \mathsf{pwd})$ and each password attempt $h'_p \leftarrow \mathcal{H}(qid, \mathcal{H}(k, \mathsf{pwd}'))$ that the device sends to the server. At a first glance that might seem unnecessary since the models already assumes secure erasures. However, secure erasures are not sufficient as an adversary can intercept the ciphertexts and later corrupt the server to learn the decryption key. It then expects all ciphertexts to open to the proper password hashes (that in the security proof might be unknown when the ciphertexts are generated). The non-committing encryption provides exactly that flexibility. To determine the correct password hashes h_p and h'_p upon server corruption different random oracle programming techniques are deployed, eventually also relying on the password guessing interfaces of the ideal functionality (if both parties are corrupted).

Similar care must also be taken for the intermediate state records that the device keeps during interactive protocols. After sending a signing request, the device cannot store the message m, or even the randomness r and the message hash $h_m \leftarrow \mathcal{H}(r, m)$, as the simulator does not learn m upon corrupting the device. Nevertheless, the device must be able to verify whether the server's contribution is correct. Therefore, when sending the message hash h_m, the device also sends a value $t \leftarrow \mathcal{H}(\mathtt{MAC}, qid, k, h_m)$ that acts as a message authentication code (MAC) for h_m. This allows the device to check that the server signed the correct message upon receiving the signature share, but without requiring state information that depends on m.

Authentication of Participants. As already mentioned earlier, the session identifier *sid* contains the identities of the device $\mathcal{D}$ and the server $\mathcal{S}$. This means that $\mathcal{D}$ and $\mathcal{S}$ have to be

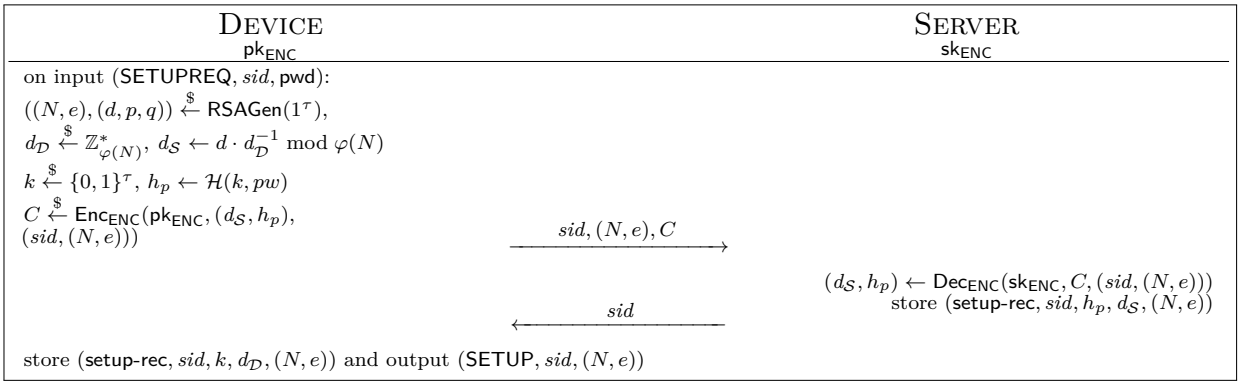

Figure 4.6: Main steps of the setup protocol for $\mathcal{F}_{\mathsf{Pass2Sign}}$

authenticated. This is achieved by employing $\mathcal{F}_{\mathsf{Auth}}$ for authenticated communication, thereby making abstraction of how the authentication is performed. This could be through a shared secret, through digital signatures (e.g., TLS with client authentication), or "offline" by letting the user use a trusted third party to register the device, such as a bank or a local municipal office. The last option has the additional advantage that one could also check the name or other credentials of the user, and also directly certify the resulting public key of the user.

4.3.1 Protocol Description

Now, the detailed protocol for the Pass2Sign scheme and a simplified presentation in Figure 4.6 and 4.7 is presented. Also, a public-key infrastructure is assumed, where devices and servers can register their public keys, modeled by the ideal functionality $\mathcal{F}_{\mathsf{CA}}$ [Can04], and authenticated message transmission, modeled by $\mathcal{F}_{\mathsf{Auth}}$. In the protocol description, inputs to and outputs from protocols are given informally to make the protocol more readable (e.g., that $\mathcal{S}$ sends m to $\mathcal{D}$ via $\mathcal{F}_{\mathsf{Auth}}$ instead of an explicit call to $\mathcal{F}_{\mathsf{Auth}}$ with sub-session IDs etc.). Further, all parties check the correctness of session and sub-session IDs in all inputs. Moreover, $\mathcal{H}$ and $\mathcal{H}_{\mathsf{RSA}}$ denote shorthand notations for two random-oracle functionalities $\mathcal{F}_{\mathsf{RO}}^{\{0,1\}^*\to\{0,1\}^\lambda}$ and $\mathcal{F}_{\mathsf{RO}}^{\{0,1\}^*\to\mathbb{Z}_N^*}$, respectively. Note that these are single-instance functionalities; one can obtain a secure multi-instance implementation by prefixing each call to them with *sid*. The protocol further makes use of an RSA-key generator RSAGen.

As discussed earlier, secure erasures are necessary to achieve the security guarantees. Thus, it is assumed that after each protocol step all variables are deleted unless explicitly state that a variable is stored. Finally, it is assumed that whenever a check performed by the server or device fails, the checking party aborts the protocol, as defined in Chapter 2.

Initialization Protocol. This step initializes the server. In particular, this interface allows the environment to say that the server generates, and registers, the key pair of the RECV-SIM-secure encryption scheme.

Step 1. Server $\mathcal{S}$ generates public key:

a) Upon input (INIT, *sid*), generate a key pair for a RECV-SIM-secure ENC: $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.

b) Create record $(\mathsf{keypair\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{S}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{S}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{S}})$.
c) Send $(\mathsf{REGISTER}, (\mathcal{D}, (sid, \mathcal{F}_{\mathsf{CA}})), \mathsf{pk}_{\mathsf{ENC},\mathcal{S}})$ to $\mathcal{F}_{\mathsf{CA}}$.

Setup Protocol. The setup procedure is the following protocol that a device $\mathcal{D}$ runs on input $(\mathsf{SETUPREQ}, sid, \mathsf{pwd})$ with server $\mathcal{S}$, where $sid = (\mathcal{S}, \mathcal{D}, sid')$.

Step 2. Device generates account data:

a) Upon input $(\mathsf{SETUPREQ}, sid, \mathsf{pwd})$, retrieve $\mathsf{pk}_{\mathsf{ENC}}$ for $\mathcal{S}$ from $\mathcal{F}_{\mathsf{CA}}$. If $\mathsf{pk}_{\mathsf{ENC}} = \perp$, ignore.
b) Generate RSA key material as $(N, e, d, p, q) \xleftarrow{\$} \mathsf{RSAGen}(1^\tau)$ and share the secret exponent d by choosing a random $d_{\mathcal{D}} \xleftarrow{\$} \mathbb{Z}^*_{\varphi(N)}$ and setting $d_{\mathcal{S}} \leftarrow d \cdot d_{\mathcal{D}}^{-1} \bmod \varphi(N)$. It is required that $d_{\mathcal{S}}$ is encoded as an $|N|$-bit string from now on.
c) Compute $h_p \leftarrow \mathcal{H}(k, \mathsf{pwd})$ for a random $k \xleftarrow{\$} \{0,1\}^\tau$.
d) Encrypt the RSA key share $d_{\mathcal{S}}$ and the authentication information h_p under $\mathsf{pk}_{\mathsf{ENC}}$ with label $(sid, (N, e))$. That is, compute $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, (d_{\mathcal{S}}, h_p), (sid, (N, e)))$.
e) Store the record $(\mathsf{setup\text{-}rec\text{-}temp}, sid, k, d_{\mathcal{D}}, (N, e))$ and send the message $m = (sid, (N, e), c)$ to the server $\mathcal{S}$ using $\mathcal{F}_{\mathsf{Auth}}$.

Step 3a. Server registers account:

a) Upon receiving $m = (sid, (N, e), c)$ from $\mathcal{D}$ via $\mathcal{F}_{\mathsf{Auth}}$, check that sid is not registered yet. Ignore otherwise.
b) Decrypt c as $(d_{\mathcal{S}}, h_p) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, (sid, (N, e)))$. If successful, store $(\mathsf{setup\text{-}rec}, sid, h_p, d_{\mathcal{S}}, (N, e))$.
c) Acknowledge the created account by sending (sid) to $\mathcal{D}$ via $\mathcal{F}_{\mathsf{Auth}}$. At this point, the server has registered an account.

Step 3b. Device completes registration:

a) Upon receiving a message (sid) from $\mathcal{S}$ via $\mathcal{F}_{\mathsf{Auth}}$, check that a record $(\mathsf{setup\text{-}rec\text{-}temp}, sid, k, d_{\mathcal{D}}, (N, e))$ for sid exists.
b) Store $(\mathsf{setup\text{-}rec}, sid, k, d_{\mathcal{D}}, (N, e))$ and end with output $(\mathsf{SETUP}, sid, (N, e))$.

Signing Protocol. This protocol starts when the device $\mathcal{D}$ receives an input $(\mathsf{SETUPREQ}, sid, qid, m, \mathsf{pwd}')$, where $sid = (\mathcal{S}, \mathcal{D}, sid')$, upon which it runs the following protocol with the server $\mathcal{S}$. Recall that it is assumed that both parties have previously agreed upon a common and globally unique query identifier qid. All messages sent between the device and server also contain the qid as prefix, and only those messages with the corresponding qid are further processed, i.e., all other requests are silently ignored.

Step 4. Device sends signing request:

a) Upon input $(\mathsf{SETUPREQ}, sid, qid, m, \mathsf{pwd}')$, look up record $(\mathsf{SETUP}, sid, k, d_{\mathcal{D}}, (N, e))$.
b) "Blind" the message by drawing $r \xleftarrow{\$} \{0,1\}^\tau$ and computing $h_m \leftarrow \mathcal{H}(r, m)$.
c) Compute the (re-)authentication value $h'_p \leftarrow \mathcal{H}(qid, \mathcal{H}(k, \mathsf{pwd}'))$.
d) Compute a "MAC" t of h_m as $t \leftarrow \mathcal{H}(\texttt{"MAC"}, qid, k, h_m)$.

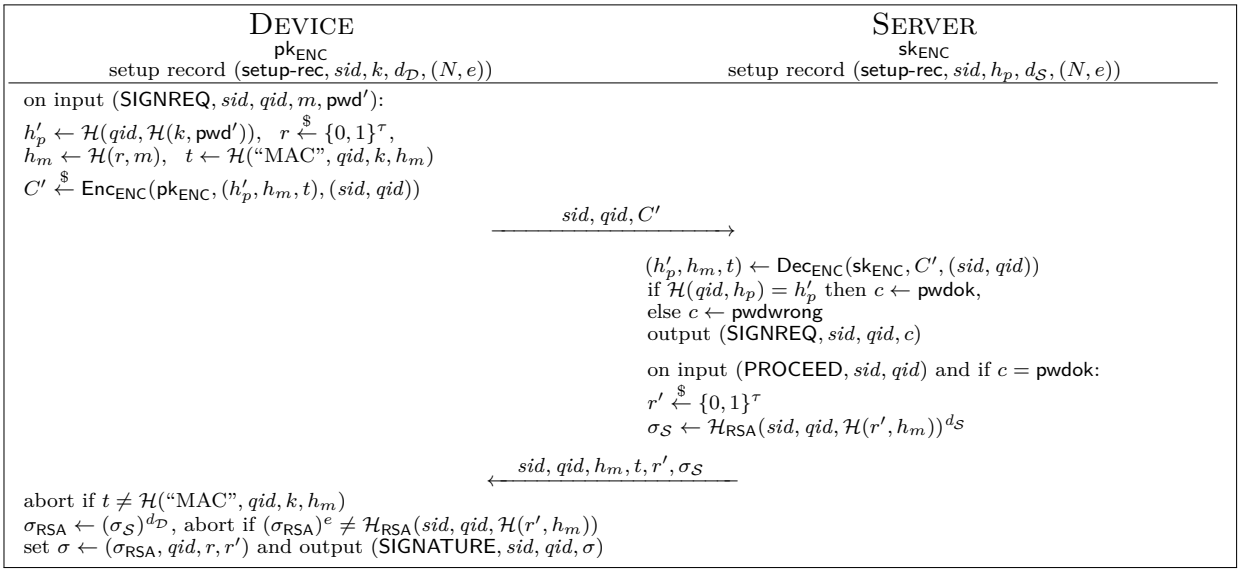

Figure 4.7: Main steps of the signing protocol for $\mathcal{F}_{\mathsf{Pass2Sign}}$

e) Generate a non-committing encryption of h'_p, h_m, and t under the public key $\mathsf{pk}_{\mathsf{ENC}}$ and with label (sid, qid) as $c' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, (h'_p, h_m, t), (sid, qid))$.
f) Store the record $(\mathsf{sign\text{-}rec}, sid, qid, r)$ and send (sid, qid, c') to $\mathcal{S}$ via $\mathcal{F}_{\mathsf{Auth}}$.

Step 5. Server verifies information:

a) Upon receiving (sid, qid, c') from $\mathcal{D}$ via $\mathcal{F}_{\mathsf{Auth}}$, retrieve $(\mathsf{SETUP}, sid, h_p, d_{\mathcal{S}}, (N, e))$ for sid.
b) Decrypt c' to $(h'_p, h_m, t) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c', (sid, qid))$.
c) Check the password by verifying whether $\mathcal{H}(qid, h_p) = h'_p$ and set $c \leftarrow \mathsf{pwdok}$ if so, and $c \leftarrow \mathsf{pwdwrong}$ otherwise.
d) Store the record $(\mathsf{sign\text{-}rec}, sid, qid, h_m, t, c)$ and output $(\mathsf{SETUPREQ}, sid, qid, c)$.

Step 6. Server creates its signature share:

a) Upon input $(\mathsf{PROCEED}, sid, qid)$, retrieve $(\mathsf{sign\text{-}rec}, sid, qid, h_m, t, c)$ for qid. Abort if $c \neq \mathsf{pwdok}$.
b) Compute the signature share $\sigma_{\mathcal{S}} \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid, \mathcal{H}(r', h_m))^{d_{\mathcal{S}}} \bmod N$ for a random $r' \xleftarrow{\$} \{0,1\}^\tau$.
c) Send $(sid, qid, h_m, t, r', \sigma_{\mathcal{S}})$ to $\mathcal{D}$ via $\mathcal{F}_{\mathsf{Auth}}$.

Step 7. Device completes the signature:

a) Upon receiving $(sid, qid, h_m, t, r', \sigma_{\mathcal{S}})$ from $\mathcal{S}$ via $\mathcal{F}_{\mathsf{Auth}}$, retrieve $(\mathsf{sign\text{-}rec}, sid, qid, r)$ for qid and setup record $(\mathsf{setup\text{-}rec}, sid, k, d_{\mathcal{D}}, (N, e))$.
b) Verify that $t = \mathcal{H}(\texttt{"MAC"}, qid, k, h_m)$.
c) Complete the signature by computing $\sigma_{\mathsf{RSA}} \leftarrow (\sigma_{\mathcal{S}})^{d_{\mathcal{D}}} \bmod N$. Verify that $(\sigma_{\mathsf{RSA}})^e = \mathcal{H}_{\mathsf{RSA}}(sid, qid, \mathcal{H}(r', h_m)) \bmod N$ holds, i.e., that the server's signature share was correct.
d) Set $\sigma \leftarrow (\sigma_{\mathsf{RSA}}, qid, r, r')$. End with output $(\mathsf{SIGNATURE}, sid, qid, \sigma)$.

Step 8. Signature Verification. On input $(\mathsf{VERIFY}, sid, m, \sigma, \mathsf{pk})$, parse $\mathsf{pk} = (N, e)$, $\sigma = (\sigma_{\mathsf{RSA}}, qid, r, r')$. Set $M \leftarrow (sid, qid, \mathcal{H}(r', \mathcal{H}(r, m)))$. If σ_{RSA} is a valid RSA signature on M, i.e., if $0 < \sigma_{\mathsf{RSA}} < N$ and $\mathcal{H}_{\mathsf{RSA}}(M) = (\sigma_{\mathsf{RSA}})^e \bmod N$, output $(\mathsf{VERIFIED}, sid, m, \sigma, \mathsf{pk}, \mathsf{true})$, and $(\mathsf{VERIFIED}, sid, m, \sigma, \mathsf{pk}, \mathsf{false})$ otherwise.

4.4 Security

Now it is proven that the given protocol is really secure in the UC-sense.

Theorem 4.5. *The* Pass2Sign *scheme described in Section 4.3 securely implements the ideal functionality* $\mathcal{F}_{\mathsf{Pass2Sign}}$ *defined in Section 4.2 in the* $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Auth}})$*-hybrid model with secure erasures if the RSA one-wayness assumption associated to* RSAGen *holds and* $\mathsf{ENC} = (\mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Dec}_{\mathsf{ENC}})$ *is an RECV-SIM secure encryption scheme.*

Using the RECV-SIM secure encryption scheme proposed in Section 4.1.3, which is an extension of the Bellare-Rogaway CCA2 encryption scheme, and instantiated with the RSA trapdoor permutation, the following corollary is obvious:

Corollary 4.6. *The* Pass2Sign *scheme described in Section 4.3 and instantiated as proposed, securely implements the ideal functionality* $\mathcal{F}_{\mathsf{Pass2Sign}}$ *defined in Section 4.2 in the* $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Auth}})$*-hybrid model with secure erasures if the RSA assumption associated with* RSAGen *holds.*

High-Level Idea. The proof of Theorem 4.5 is done by providing a simulator and a sequence of games. The initial game is the real experiment and the final game (Game 10) runs the simulator given only the information that is also available to the adversary when interacting with the ideal functionality. Next, a rough sketch of the game sequence is given. The detailed description as well as the description of the simulator is given afterwards.

Game 1 and Game 2 abort when collisions occur in random-oracle outputs, or if the adversary "predicts" random-oracle outputs. Game 3 replaces all ciphertexts sent by an honest device to an honest server with simulated "dummy" ciphertexts. It uses the decryption simulation to decrypt ciphertexts that were not sent by the honest device, and uses the key-leakage simulation to obtain the secret key $\mathsf{sk}_{\mathsf{ENC}}$ in case the server is corrupted.

In Game 4 and Game 5, the simulator aborts when a valid signature is verified that did not originate from the device or the server, whichever is still honest. Interestingly, the case for an honest device can be reduced from the unforgeability of RSA-FDH, but for an honest server it has to be reduced straight from the RSA assumption (similar as in [BS01a]). Thus, the standard RSA assumption in the random-oracle model is enough in this case.

The subsequent games are all about making the setup and signing protocol simulations independent of the actual values of the passwords and messages being signed. Game 6 and Game 7 make the setup and signing protocol simulations independent of the actual value of the password and password attempts. When the server is corrupt, instead of deriving h_p and t from the device secret k and the real passwords, the simulator uses random values. For $h'_{p,i}$ it either uses $\mathcal{H}(qid, h_p)$ or a random value, depending whether the password attempt used in a signing request is correct or not. As long as the device is honest, this change cannot be detected by the adversary as he does not know k. When the device gets corrupted too, and thus the

adversary learns k, the simulator starts programming the random oracle consistently to the previously chosen h_p and $h'_{p,i}$ whenever the actual passwords pwd and pwd'_i are queried. In the final simulation with the ideal functionality this is done with the help of the password guessing interface that becomes available as soon as both, the device and the server, are corrupted.

If both entities are initially honest and the device gets corrupted first, the simulation is different though. Then, one needs to keep $h_p, h'_{p,i}$ unassigned and only fix their values when the server gets corrupted too. This is sufficient as the device never stores the hash values and sends them only in encrypted form to the server (which are dummy ciphertexts since Game 3). However, as soon as the server gets corrupted, the adversary learns the secret key of the encryption scheme and thus all previous dummy communication between the device (sent by the simulator when $\mathcal{D}$ was still honest) and the server must decrypt to the correct hash values. Here, one cannot assign random values to $h_p, h'_{p,i}$, though. This stems from the fact that the device got corrupted first, and thus the adversary knows the device secret k and could have already computed h_p or $h'_{p,i}$ for the correct password values $\mathsf{pwd}, \mathsf{pwd}_i$. Thus, in order to ensure consistency, check whether a previously answered random oracle query contained an actual password pwd or pwd'_i (when switching to the ideal world, this is done via the password guessing interface). If such a query is found, simply reuse the previously given random oracle response for h_p or $h'_{p,i}$ and assign random values otherwise. Then, use the keyleak-simulator of the non-committing encryption scheme to obtain a key $\mathsf{sk}_{\mathsf{ENC}}$ that decrypts the dummy ciphertexts from Game 3 to the just determined hash values. In fact, this case is the reason why one needs *non-committing* encryption, as in the ideal world the passwords pwd or pwd'_i are unknown to the simulator (even when the device gets corrupted), and the password guessing interface only becomes available when both entities got corrupted.

Game 8 and Game 9 make the signing process independent of the message m, with an interesting technique that programs the relevant random-oracle entries only when the signature is verified, not when it is created. In Game 8, an honest device interacting with a corrupt server chooses random values r and h_m and, if the server behaves honestly, computes the signature as $\sigma_{\mathsf{RSA}} = \mathcal{H}_{\mathsf{RSA}}(sid, qid, \mathcal{H}(r', h_m))^{d_{\mathcal{S}} d_{\mathcal{D}}} \bmod N$, which it can compute because it knows the keys $d_{\mathcal{S}}$ and $d_{\mathcal{D}}$ from the time they were generated. The simulator stores the resulting signature $\sigma = (\sigma_{\mathsf{RSA}}, qid, r, r')$ as valid for m in its records; in the ideal world, the simulator would input $(\mathsf{SIGNATURE}, sid, qid, \sigma)$ to let the functionality associate σ to m. When later a random-oracle query $\mathcal{H}(r, m)$ comes in, the simulator looks up whether a signature σ was recorded with randomness r, and if so, checks whether σ has been recorded as a valid signature for m (using its own records, or using the VERIFY interface in the ideal world). If so, then it uses the value h_m used during the corresponding signing protocol as the random-oracle response, otherwise it returns a random value. Game 9 acts similarly when the device and server are both honest, but by letting the server sign random h'_m values, without knowing the corresponding h_m. Only when a query $\mathcal{H}(r', h_m)$ is made, h'_m is assigned if a corresponding signature has been recorded.

4.4.1 Detailed Sequence of Games

The proof consist of a sequence of games that a challenger runs with the real-world adversary. In the final game the transition to let the challenger run internally the ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$ and simulate all messages based merely on the information it can obtain from $\mathcal{F}_{\mathsf{Pass2Sign}}$ is made. Now, each Game i is described, while it is also argued why the view of the environment does

not significantly change.

Game 0: In the first game, the challenger executes the real protocol for *all* honest players, obtaining their inputs from, and passing the respective outputs to the environment.

Game 1: Let $\mathcal{T}_{\mathcal{H}}$ be the table in which the simulator stores query-response pairs (m, h) when simulating random-oracle queries to $\mathcal{H}$. Recall that all random oracle calls to $\mathcal{H}$ are implicitly prefixed with *sid* of the current account. Similarly, In this hop, a table $\mathcal{T}_{\mathcal{H}}$ is created per *sid*. The explicit handling of the *sid* is omitted here as well. This game aborts whenever the adversary or the challenger cause a collision in $\mathcal{H}$ or $\mathcal{H}_{\mathsf{RSA}}$, or the adversary sends a "correctly predicted" hash value. A "correctly predicted" value is some hash h_p, h'_p, h_m that neither resulted from a query to the random oracle nor was previously generated by the challenger, and the adversary makes a random-oracle query that maps to this value only after having sent the hash.

By the random choice of the response from $\{0,1\}^\lambda$, the adversary can distinguish this game hop only with negligible probability.

Game 2: From now on, the challenger creates additional internal records for setup and signing. When setup is done by an honest device, it stores $(\mathsf{setup\text{-}rec}, sid, k, h_p, d_{\mathcal{S}}, d_{\mathcal{D}}, (N, e))$, i.e., it also keeps the RSA key share $d_{\mathcal{S}}$ of the server. For an account created by a corrupt device with an honest server, the challenger maintains $(\mathsf{setup\text{-}rec}, sid, \perp_k, h_p, d_{\mathcal{S}}, \perp_{\mathcal{S}}, (N, e))$.

Similarly, when an honest device starts a signing request, the challenger initiates a record $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, m_i, t_i, r_i, \perp_{r'}, \perp_\sigma)$. For a signing session between a corrupt device and honest server, the challenger creates $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \perp_m, t_i, \perp_r, r', \perp_\sigma)$. Both records will be completed with the missing values as soon as they are generated or received by an honest party.

Clearly, this is only an internal change and has no effect on the view of the environment, i.e., the views are equal.

Game 3: In this game, every non-committing encryption (via a series of hybrids) is replaced that would be sent between two honest parties by a simulated ciphertext. More precisely, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ of the NCE scheme in mode $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{PUBLICKEY}, 1^\lambda)$ is run to obtain the public key $\mathsf{pk}_{\mathsf{ENC}}$ that the honest server registers with $\mathcal{F}_{\mathsf{CA}}$. Then, whenever the honest device has to create an encryption of m with label ℓ under $\mathsf{pk}_{\mathsf{ENC}}$ the real ciphertext is replaced by $C \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPT}, |m|, \ell)$ where $|m|$ denotes the message length. Also, an internally maintained list $\mathcal{Q}$ of tuples (C, m, ℓ) mapping the real messages to the "dummy" ciphertexts is deployed.

When an honest server receives such a simulated ciphertext, it does not decrypt C but looks up the corresponding plaintext from its internal record, i.e., either $\mathsf{setup\text{-}rec}$ for ciphertexts received in setup and $\mathsf{sign\text{-}rec}$ for ciphertexts received in the signing protocol. If the server receives a ciphertext/label pair (C', ℓ') where $(C', \cdot, \ell') \notin \mathcal{Q}$, i.e., that was not created by an honest device, $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{DECRYPT}, C', \ell')$ is run instead of the decryption algorithm.

When the server eventually gets corrupted, $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$ is finally run to obtain the secret decryption key $\mathsf{sk}_{\mathsf{ENC}}$ which has to be handed to the adversary.

This game hop is indistinguishable by the RECV-SIM security of the encryption scheme ENC. A reduction is straightforward and thus omitted.

Game 4: Here, it is aborted if an honest party obtains a (valid) input $(\mathsf{VERIFY}, sid, m^*, \sigma^*, \mathsf{pk})$ for a public key pk of an honest device, but for a signature σ^* that the device has never produced. Receiving such a forged signature in the protocol allows to construct an adversary $\mathcal{B}$ that breaks the strong unforgeability of RSA-FDH signatures with non-negligible probability. For the sake of simplicity, the same reduction for the case of an honest and a corrupt server is used.

When $\mathcal{B}$ receives the challenge RSA public key $\mathsf{pk} = (N, e)$, choose $d_{\mathcal{S}}$ at random from $\mathbb{Z}_N$, leave $d_{\mathcal{D}}$ unassigned, and normally compute the authentication values for the protocol. Recall that the challenger also internally stores $d_{\mathcal{S}}$. For each signing query m_i, the honest device chooses random r_i and h_{m_i}, gets $h_{m_i} \leftarrow \mathcal{H}(r_i, m_i)$ from the random oracle, and sends h_{m_i} to the server. When it receives a response $(sid, qid_i, h_{m,i}, t_i, r'_i, \sigma_{\mathcal{S},i})$, $\mathcal{B}$ verifies that $\sigma_{\mathcal{S},i} = \mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m_i}))^{d_{\mathcal{S}}}$ or aborts otherwise. Then $\mathcal{B}$ sends $M_i \leftarrow (sid, qid_i, \mathcal{H}(r'_i, h_{m_i}))$ to its RSA-signing oracle to obtain the signature $\sigma_{\mathsf{RSA},i}$. The device outputs $(\mathsf{SIGNATURE}, sid, qid_i, \sigma_i)$, where $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$. Each store each produced message/signature tuple (m_i, σ_i) in a list $\mathcal{L}_\sigma$.

When $\mathcal{B}$ receives input $(\mathsf{VERIFY}, sid, m^*, \sigma^*, \mathsf{pk})$ such that $(m^*, \sigma^*) \notin \mathcal{L}_\sigma$, it parses σ^* as $(\sigma^*_{\mathsf{RSA}}, qid^*, r^*, r'^*)$ and outputs $(M^*, \sigma^*_{\mathsf{RSA}})$ as forgery with $M^* \leftarrow (sid, qid^*, \mathcal{H}(r'^*, \mathcal{H}(r^*, m^*)))$. The output is a valid RSA-FDH forgery if the message/signature tuple $(M^*, \sigma^*_{\mathsf{RSA}})$ did not appear as an RSA signing query/response before. Suppose for contradiction that it did appear in a signing query by $\mathcal{B}$, i.e., $M^* = (sid, qid^*, \mathcal{H}(r'^*, \mathcal{H}(r^*, m^*))) = M_i = (sid, qid_i, \mathcal{H}(r'_i, \mathcal{H}(r_i, m_i)))$ for some query i that led to signature σ^*_{RSA}. That would mean that $qid^* = qid_i$ and $\mathcal{H}(r'^*, \mathcal{H}(r^*, m^*)) = \mathcal{H}(r'_i, \mathcal{H}(r_i, m_i))$. As collisions on $\mathcal{H}$ were excluded in Game 1, the latter means that $r'^* = r'_i$, $r^* = r_i$, and $m^* = m_i$. This contradicts the fact that $(m^*, \sigma^*) \notin \mathcal{L}_\sigma$.

Note that the simulation deviated from how the signing key $d_{\mathcal{S}}$ for the server was chosen. In the original scheme, $d_{\mathcal{S}}$ is chosen from $\mathbb{Z}^*_{\varphi(N)}$, whereas $d_{\mathcal{S}}$ was selected randomly from $\mathbb{Z}_N$, since the challenger only learns (N, e). However, Lemma 5.1 from [BS01a] shows that $\Pr[d_{\mathcal{S}} \in \mathbb{Z}^*_{\varphi(N)}] > \frac{8}{435 \ln |N|}$ for RSA modulus N and $d_{\mathcal{S}} \xleftarrow{\$} \mathbb{Z}_N$. Hence, a "good" $d_{\mathcal{S}}$ value was chosen with non-negligible probability, in which case the simulation of the protocol was perfect. Overall, this game hop is indistinguishable by the strong unforgeability of the signature scheme $\mathsf{DSIG}_{\mathsf{RSA}}$.

Game 5: This game considers the setting where the server is honest and the device is corrupt, but was honest during setup. Then, similar to the previous game, but there is an abort if there is a forged signature for the corresponding key pk. That is, an abort happens if there is a valid signature (m^*, σ^*) where $\sigma^* = (\sigma^*_{\mathsf{RSA}}, qid^*, r^*, r'^*)$ but the honest server never signed $M^* = (sid, qid^*, h'_m)$ where $h'^*_m = \mathcal{H}(r'^*, \mathcal{H}(r^*, m^*))$ in the session qid^*, or there are two different valid message-signature pairs $(m^*_1, \sigma^*_1) \neq (m^*_2, \sigma^*_2)$ that resulted from the same signing protocol for $M^* = (sid, qid^*, h'_m)$. In contrast to the previous game, this cannot be reduced to the unforgeability of the RSA signature as underlying assumption, though, as the honest server does not have "full" control of the final signature. However, a forgery in the scheme allows to break the RSA problem in the random-oracle model directly.

The latter case of two different message-signature pairs either requires the adversary to find collisions in the random-oracle responses for $\mathcal{H}$ and $\mathcal{H}_{\mathsf{RSA}}$, which were excluded in GAME 1, or to find two different values $0 < \sigma^*_{\mathsf{RSA},1} < \sigma^*_{\mathsf{RSA},2} < N$ such that $\mathcal{H}_{\mathsf{RSA}}(M^*) \equiv {\sigma^*_{\mathsf{RSA},1}}^e \equiv {\sigma^*_{\mathsf{RSA},2}}^e \bmod N$. If $\mathcal{H}_{\mathsf{RSA}}(M^*) \in \mathbb{Z}_N^*$, then this is impossible because RSA is a permutation on $\mathbb{Z}_N^*$. If $\mathcal{H}_{\mathsf{RSA}}(M^*) \notin \mathbb{Z}_N^*$, then $\gcd(N, \mathcal{H}_{\mathsf{RSA}}(M^*)) > 1$, so that $\mathcal{B}$ can factor N and solve the RSA problem.

For the former case of a (non-strong) forgery against the protocol, consider algorithm $\mathcal{B}$ that, on input an RSA public key N, e and challenge $y \in \mathbb{Z}_N^*$, outputs x where $x^e \equiv y \bmod N$ with non-negligible probability. $\mathcal{B}$ chooses keys $d_{\mathcal{D}} \xleftarrow{\$} \mathbb{Z}_N$ and $k \xleftarrow{\$} \{0,1\}^\lambda$. When the device gets corrupted, $\mathcal{B}$ hands these values to the adversary. When the adversary makes a random-oracle query $\mathcal{H}_{\mathsf{RSA}}(M_i)$, $\mathcal{B}$ chooses $x_i \xleftarrow{\$} \mathbb{Z}_N^*$ and responds $x_i^e y \bmod N$. When the honest server receives an incoming signing request for message hash h_{m_i}, it chooses $r'_i \xleftarrow{\$} \{0,1\}^\lambda$, $\sigma_{\mathcal{S}_i} \xleftarrow{\$} \mathbb{Z}_N^*$, programs $\mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m_i})) = \sigma_{\mathcal{S}_i}^{e d_{\mathcal{D}}} \bmod N$, adds $(h_{m_i}, \sigma_{\mathcal{S}_i}^{d_{\mathcal{D}}})$ to $\mathcal{L}_\sigma$, and sends $\sigma_{\mathcal{S}_i}$ back to the device. In case $\mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m_i}))$ had been queried before, $\mathcal{B}$ aborts, but by the random choice of r'_i, this happens with probability at most $\frac{q_s q_h}{2^\lambda}$, where q_s and q_h are the number of signing and random-oracle queries made by the adversary, respectively. When $\mathcal{B}$ receives an input $(\mathsf{VERIFY}, sid, m^*, (\sigma^*_{\mathsf{RSA}}, qid^*, r^*, r'^*), \mathsf{pk})$ such that $(\mathcal{H}(r^*, m^*), \sigma^*) \notin \mathcal{L}_\sigma$, $\mathcal{B}$ looks in its random-oracle responses to find x^* such that $\mathcal{H}_{\mathsf{RSA}}(sid, qid^*, \mathcal{H}(r'^*, \mathcal{H}(r^*, m^*))) = x^{*e} y \bmod N$, then by the validity of the signature it holds that ${\sigma^*_{\mathsf{RSA}}}^e \equiv x^{*e} y \bmod N$, so $\mathcal{B}$ returns $\sigma^*_{\mathsf{RSA}}/x^* \bmod N$ as the RSA inversion of y. As in the previous game, Lemma 5.1 from [BS01a] is a lower bound on the probability that $d_{\mathcal{D}} \in \mathbb{Z}^*_{\varphi(N)}$.

Game 6: Now, all values that would depend on the device secret k are made independent of k, when they are generated by an honest device and are sent to a corrupt server. More precisely, the way the authentication values $h_p, h'_{p,i}$ and the tags t_i are computed is changed. Namely, $h_p \xleftarrow{\$} \{0,1\}^\lambda$ is chosen randomly at setup at setup, while the actual password pwd is stored in an internal record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd})$. For a sign request started by an honest device $h'_{p,i} \leftarrow \mathcal{H}(qid_i, h_p)$ is derived, if the request was initiated for a password $\mathsf{pwd}'_i = \mathsf{pwd}$ and set it to a random value $h'_{p,i} \xleftarrow{\$} \{0,1\}^\lambda$ if the passwords did not match. The created $h'_{p,i}$ are kept in the $\mathsf{sign\text{-}rec}$ record as before, while also another record for the password attempts as $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}'_i)$ is created.

The computation of the tag $t_i \leftarrow \mathcal{H}(\texttt{"MAC"}, qid_i, k, h_{m,i})$ is replaced by $t_i \xleftarrow{\$} \{0,1\}^\lambda$, and t_i is stored in $\mathsf{sign\text{-}rec}$ as before. Also, the verification of the tag in Step 4 of the signing protocol is replaced by a simple look up whether the received t_i value is the same as in $\mathsf{sign\text{-}rec}$.

If there is a random oracle query of the form $\mathcal{H}(k, \cdot)$ or $\mathcal{H}(\texttt{"MAC"}, \cdot, k, \cdot)$ for the secret key k that was chosen and the device is still honest, abort. By the random choice of $k \xleftarrow{\$} \{0,1\}^\lambda$, and the fact that all values were made independent of k, the probability for such a query (and the resulting abort) is negligible.

However, that behavior is changed as soon as the device gets corrupted, as then the adversary learns k and the above argument no longer holds. With the help of the internal records, a fresh k is chosen at the moment the device gets corrupted and ensure consistency with the previously chosen values $h_p, h'_{p,i}$ and t_i as follows. For a query $\mathcal{H}(k, \mathsf{pwd})$ where a record $(\mathsf{setup\text{-}rec}, sid, k, h_p, d_{\mathcal{S}}, d_{\mathcal{D}}, (N, e))$ for k and a record $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd})$ for sid, pwd

exists, set the random oracle response to be h_p (taken from the setup-rec record). Similarly, for a query $\mathcal{H}(qid_i, h_p^*)$ where $((k, \mathsf{pwd}_i'), h_p^*) \in \mathcal{T}_\mathcal{H}$, i.e., h_p^* is the result of a previous query (k, pwd_i') and records $(\mathsf{setup\text{-}rec}, sid, k, h_p, d_\mathcal{S}, d_\mathcal{D}, (N, e))$ for k, and $(\mathsf{sign\text{-}rec}, sid, qid_i, h_{p,i}', h_{m,i}, m_i, t_i, r_i, \{r_i', \perp_{r'}\}, \{\sigma_{\mathsf{RSA},i}, \perp_\sigma\})$ and $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}_i')$ for $sid, qid_i, \mathsf{pwd}_i'$ exists, the response is set to $\mathcal{H}(qid_i, h_p^*) \leftarrow h_{p,i}'$ (taken from sign-rec) and to a random value otherwise. For queries $\mathcal{H}(\texttt{"MAC"}, qid_i, k, h_{m,i})$ where a setup-rec record for k and a record $(\mathsf{sign\text{-}rec}, sid, qid_i, h_{p,i}', h_{m,i}, m_i, t_i, r_i, \{r_i', \perp_{r'}\}, \{\sigma_{\mathsf{RSA},i}, \perp_\sigma\})$ for $sid, qid_i, h_{m,i}$ exists, the random oracle response is set to the stored t_i and to a random string otherwise. Overall, this game hop is indistinguishable by the random choice of k.

Game 7: Now, a similar change to the computation of h_p and $h_{p,i}', t_i$ for the setting where a setup or sign request is done between an honest device and *honest* server is made. However, here all values h_p and $h_{p,i}', t_i$ are left unassigned at the beginning and only records $(\mathsf{setup\text{-}rec}, sid, \mathsf{pwd})$ for setup and $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}_i')$ for each signing request are created.

When the honest server is supposed to output $(\mathsf{SIGNREQ}, sid, qid_i, c_i)$ where c_i indicates whether the password attempt was successful, it determines c_i based on the passwords stored in the signreq-rec and setup-rec records.

Then, if the server gets the "ok" to proceed and the password attempt was correct, i.e., $c_i = \mathsf{pwdok}$ a random tag $t_i \xleftarrow{\$} \{0,1\}^\lambda$ is chosen and stored in the sign-rec record. The honest server then proceeds normally. When the honest device receives a message $(sid, qid_i, h_{m,i}, t_i, r_i', \sigma_{\mathcal{S},i})$ from the honest server, it only continues when it receives the same t_i that is contained in the sign-rec record.

The challenger then maintains incomplete records $(\mathsf{setup\text{-}rec}, sid, k, \perp_{h,\mathsf{pwd}}, d_\mathcal{S}, d_\mathcal{D}, (N, e))$ and $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, h_{m,i}, m_i, \{t_i, \perp_\mathsf{tag}\}, r_i, \{r_i', \perp_{r'}\}, \{\sigma_{\mathsf{RSA},i}, \perp_\sigma\})$ where t_i is only assigned when the honest server sends its signature share. This event might not occur though, e.g., because the signing request never arrived or the environment did not give the "ok" to proceed. Similarly, also the list $\mathcal{Q}$ only contains incomplete entries $(C', (\perp_{h',\mathsf{pwd}}, \perp_{h_m}, \{t_i, \perp_\mathsf{tag}\}), (sid, qid_i))$. Those tuples are fully completeed and finally assigned values to h_p and $h_{p,i}'$ as soon as the server gets corrupted.

When the server gets corrupted, and the device is still honest, h_p and t_i are chosen at random, whereas $h_{p,i}'$ is determined based on c_i. That is, if $c_i = \mathsf{pwdok}$ then $h_{p,i}' \leftarrow \mathcal{H}(qid_i, h_p)$ and $h_{p,i}' \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. The rest of the simulation for this setting is then equivalent to the previous game with a corrupt from the beginning server. Thus, in the remainder of this game focuses on the setting where the device gets corrupted first.

If the device gets corrupted (and the server is still honest), the adversary learns k and r_i, and thus is now able to compute the hashes h_p, h_p' and tag t_i himself. However, it has not learned the choices for those values yet, since the server is still honest. And in fact, still no values are assigned for h_p, h_p' or t_i (for the open signing requests).

As in the game above, it is aborted if there is a random oracle query of the form $\mathcal{H}(k, \cdot)$ or $\mathcal{H}(\texttt{"MAC"}, \cdot, k, \cdot)$ where k is the secret device key and the device is still honest. Again, as k was not used in any computation so far, such a query can only occur with negligible probability. Then, as soon as the device gets corrupted and the adversary learns k, do no longer abort for those queries. The procedure to answer queries of the form $\mathcal{H}(\texttt{"MAC"}, qid_i, k, h_{m,i})$ is the same as in the game described above. Whereas for fresh queries of the form $\mathcal{H}(k, \mathsf{pwd})$ or $\mathcal{H}(qid_i, h_p^*)$

that might be an attempt to (re)-compute h_p, h'_p simply assign random values.

If the device gets corrupted while the server is still honest, the way the server verifies whether the provided authentication information for a new signing request is correct is changed. For each such signing request (sid, qid_i, C'_i) coming from a corrupted (but initially honest) device, the honest server first decrypts $(h'_{p,i}, h_{m,i}, t_i) \xleftarrow{\$} \mathsf{SIM_{NCE}}(\mathsf{DECRYPT}, C'_i, (sid, qid_i))$ using the simulator $\mathsf{SIM_{NCE}}$ of the non-committing encryption scheme. Then, the challenger checks if the random oracle table $\mathcal{T}_{\mathcal{H}}$ contains a preimage (qid_i, h_p) for $h'_{p,i}$ and a preimage (k, pwd) for the retrieved h_p and having the correct device key k as prefix (k is stored in setup-rec for sid). If such a preimage exists, a record $(\mathsf{signreq\text{-}rec}, sid, qid_i, \mathsf{pwd}'_i)$ with $\mathsf{pwd}'_i \leftarrow \mathsf{pwd}$ and verify the correctness of the password based on the signreq-rec and setup-rec records is created. The rest of the signing protocol is handled as with a device that was corrupt from the beginning. When eventually also the server gets corrupted, the adversary learns the key $\mathsf{sk_{ENC}}$ to decrypt the communication towards the server and thus could now check if the "correct" h_p, h'_p values before were chosen. Thus, this is the moment those hashes are determined.

To this end, the challenger checks if it already responded to random oracle queries $\mathcal{H}(k, \mathsf{pwd})$ or $\mathcal{H}(qid_i, h^*_p)$ where $((k, \mathsf{pwd}'_i), h^*_p) \in \mathcal{T}_{\mathcal{H}}$, for the passwords pwd and pwd'_i stored in the setup-rec and signreq-rec records. If such queries are found, the challenger sets the value for $h_p, h'_{p,i}$ to be the random oracle answers it had given earlier. If no queries of that form were made, it assigns random hash values. Similarly, the tags t_i of incomplete signature requests are either chosen at random or by reusing the response for a query $\mathcal{H}(\texttt{"MAC"}, qid_i, k, h_{m,i})$ the adversary has made earlier. The challenger also updates its setup-rec and sign-rec records and the tuples in $\mathcal{Q}$ to contain the determined values for $h_p, h'_{p,i}$, and t_i and finally derives and outputs the decryption key $\mathsf{sk_{ENC}} \xleftarrow{\$} \mathsf{SIM_{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$. Note, at this point $\mathcal{Q}$ is fully determined, and thus $\mathsf{SIM_{NCE}}$ has all the input it requires to program the ciphertexts correctly.

Again, by the random choice of $k \xleftarrow{\$} \{0,1\}^\lambda$, the environment can distinguish this game hop only with negligible probability, due to the birthday paradox.

Game 8: In this game, the signing procedure when done between an honest device and corrupt server is modified. Roughly, the goal is to make the signature somewhat independent of m_i, in the sense that m_i is not used in protocol simulation or in the sign-rec record, but registered in a signature-rec record at the end. The connection to the real message m_i is only established when the adversary tries to verify the signature and therefore makes a random oracle query (r_i, m_i).

To this end, when an honest device wants to sign a message m_i, r_i, $h_{m,i}$ are chosen at random from $\{0,1\}^\lambda$, while all values except m_i associated with the jointly computed signature $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$ are stored, where r'_i was provided by the server. That is, the challenger then has a record $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \perp_m, t_i, r_i, r'_i, \sigma_{\mathsf{RSA},i})$ for the signing process and another one for the completed signature as $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$.

If the device did not receive the server's contribution $(\sigma_{\mathcal{S},i}, r'_i)$, the challenger only has a record $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \perp_m, t_i, r_i, \perp_{r'}, \perp_\sigma)$. However, it will fill in the missing r'_i and the signature itself as soon as the device gets corrupted. More precisely, the challenger chooses a random r'_i and computes $\sigma_{\mathsf{RSA},i} \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m,i}))^{d_{\mathcal{S}} d_{\mathcal{D}}}$ using the knowledge of $d_{\mathcal{D}}$ *and* $d_{\mathcal{S}}$ as the setup was done by an honest device. It then updates the sign-rec record accordingly and also creates the corresponding signature-rec record.

So far, each signature is now independent of m_i, as only a random $h_{m,i}$ value was signed and the random oracle was not programed to map (r_i, m_i) to $h_{m,i}$ yet. The challenger then "fixes" this when the corresponding message (r_i, m_i) is queried to $\mathcal{H}$ by the adversary using the sign-rec and signature-rec records.

That is, whenever the challenger receives a random oracle query of the form (r_i, m_i) it first checks whether it has a matching record $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \perp_m, t_i, r_i, \perp_{r'}, \perp_\sigma)$ containing the same qid_i, r_i. If it has such a record but $r'_i = \sigma_{\mathsf{RSA},i} = \perp$, the challenger aborts. Note, this case only occurs if the server never provided its contribution and the device is still honest, which in turn means the adversary did not learn r_i so far but (at most) the random value $h_{m,i}$. Due to the choice of $r_i \xleftarrow{\$} \{0,1\}^\lambda$, a query $\mathcal{H}(r_i, m_i)$ for the same r_i can only appear with negligible probability.

If the sign-rec record was completed though (except of m_i), and a record $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$ for the queried message m_i and with $\sigma_i = (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$ exists, the challenger responds by programming $\mathcal{H}(r_i, m_i)$ to $h_{m,i}$ using the previously chosen $h_{m,i}$ value from the sign-rec record. When no matching signature-rec record exists, the challenger simply sets the random oracle response to a random value.

The "just-in-time" programming of the RO is not possible if a record $((r_i, m_i), h^*) \in \mathcal{T}_\mathcal{H}$ with $h^* \neq h_{m,i}$ already exists, i.e., the adversary had "predicted" the same r_i in a random oracle query before the honest device has randomly chosen r_i from $\{0,1\}^\lambda$ during a signing request. However, given that the adversary makes at most q_h queries to the random oracle and the honest device participates in at most q_s signing sessions, the adversary has an advantage of at most $\frac{q_h q_s}{2^\lambda}$ to hit that event. Thus, the environment can distinguish that game hop with negligible probability only.

Game 9: Now, a similar change as in the previous game is done, but for the setting where an honest device runs a signing protocol with an *honest* server. The difference to the procedure above is that the device here only draws a random r_i, while $h_{m,i}$ gets chosen at the moment when the honest server creates its signature share. (Recall that since Game 3 the honest device only sends a simulated ciphertext to the server and Step 2 of the signing protocol is done solely based on the internal record maintained by the challenger.)

Thus, after completing Step 1 of the signing protocol, the challenger only maintains a record $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, \perp_{h_m}, \perp_m, \perp_{\mathsf{tag}}, r_i, \perp_{r'}, \perp_\sigma)$ (the way $h'_{p,i}$ and t_i are computed was already changed in Game 7). When the server then gets the "ok" to proceed and wants to create its signature share, it chooses a random $h_{m,i}$ and does the rest according to the protocol. The challenger also updates its record to $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, h_{m,i}, \perp_m, t_i, r_i, r'_i, \perp_\sigma)$ where t_i is determined as described in Game 7. If the honest device now receives a message $(sid, qid_i, h_{m,i}, t_i, r'_i, \sigma_{\mathcal{S},i})$ from the honest server, it further completes the sign-rec record to $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, h_{m,i}, \perp_m, t_i, r_i, r'_i, \sigma_{\mathsf{RSA},i})$ and also creates record $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$.

As in the game above, the completed sign-rec and signature-rec records are used to consistently answer random oracle queries (r_i, m_i), but abort if such a query is made but only an incomplete signing record exists and the device is still honest.

Similar as in the game above, now the behavior when the device gets corrupted and also complete the records of interrupted or discontinued signing sessions is changed. However, there is the additional case that the server might still be honest when the device got corrupted. Thus, the completion and random-oracle handling depend on the status of the adaptive corruption.

In any case, if the device gets corrupted, r_i values of all signing requests must be provided, including ongoing ones, to the adversary. That is, the adversary now knows the randomness that was allegedly used to compute the (possibly still unassigned) $h_{m,i}$ value. Thus, the signing records of interrupted signing sessions are completed such that the "just-in-time" random oracle programming for queries (r_i, m_i) is possible, as in the previous game. However, if the server is still honest only records of the form $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, h_{m,i}, \perp_m, t_i, r_i, r'_i, \perp_\sigma)$ are completed, i.e., where the honest server had already sent its signature share. For those, the challenger computes the full signature and creates a record $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$.

In difference to the previous game that handled the setting of an (initially) honest device and corrupt server, also to random oracle queries (r_i, m_i) for which only a record $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h',\mathsf{pwd}}, \perp_{h_m}, \perp_m, \perp_{\mathsf{tag}}, r_i, \perp_{r'}, \perp_\sigma)$ exists is responded. If such a query occurs, the challenger responds with a random $h_{m,i}$ and adds $((r_i, m_i), h_{m_i})$ to $\mathcal{T}_\mathcal{H}$. Those rather empty sign-rec records are completed at the moment when both, the device and server, are corrupted.

That is, as soon as the server gets corrupted (too), the challenger completes those signatures, but by signing a random $h'_{m,i}$ and choosing a random r'_i. That is, $h'_{m,i}$ is not a "proper" random oracle response yet. The challenger also updates its sign-rec record to include $h'_{m,i}, r'_i, \sigma_{\mathsf{RSA},i}$ and creates a full signature record. Thus, while $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$ is complete now, the record $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, \perp_{h_m}, \perp_m, t_i, r_i, r'_i, \sigma_{\mathsf{RSA},i})$ still misses the $h_{m,i}$ value. (The way $h'_{p,i}$ is chosen is described in Game 7.)

The challenger then determines the "correct" value for $h_{m,i}$ by going through all answered random oracle queries that have the form $((r_i, m_i), h_{m,i}) \in \mathcal{T}_\mathcal{H}$. For each such entry the challenger checks if a matching signature record $(\mathsf{signature\text{-}rec}, sid, qid_i, m_i, \sigma_i)$ exists, i.e., the record contains the same m_i and $\sigma_i = (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$ contains r_i. If that is the case, the challenger now fully completes the sign-rec record by including $h_{m,i}$ (taken from $\mathcal{T}_\mathcal{H}$) and m_i (taken from signature-rec). It also sets $\mathcal{H}(r'_i, h_{m,i}) \leftarrow h'_{m,i}$, i.e., it links $h_{m,i}$ to the random hash value $h'_{m,i}$ it has signed. If there is no such matching random oracle query $((r_i, m_i), h_{m,i}) \in \mathcal{T}_\mathcal{H}$, the challenger chooses a random $h_{m,i} \xleftarrow{\$} \{0,1\}^\lambda$, sets $\mathcal{H}(r'_i, h_{m,i}) \leftarrow h'_{m,i}$ and also updates its sign-rec record to contain $h_{m,i}$. Thus, all missing values $h_{m,i}$ get assigned as soon as the server gets corrupted. The challenger then uses those sign-rec records to complete all tuples $(C', (h'_{p,i}, h_{m,i}, t_i), (sid, qid_i)$ for $\mathcal{Q}$. All pairs are needed to get $\mathsf{sk}_\mathsf{ENC} \xleftarrow{\$} \mathsf{SIM}_\mathsf{NCE}(\mathsf{KEYLEAK}, \mathcal{Q})$.

From now on, every new query $\mathcal{H}(r_i, m_i)$ is answered as in the previous game, i.e., by checking if a corresponding sign-rec record exists, in which case the random oracle response is set to $h_{m,i}$ contained in sign-rec. Again, such programming fails if the adversary queried $\mathcal{H}(r_i, \cdot)$ or $\mathcal{H}(r'_i, \cdot)$ and the challenger has subsequently chosen the same r_i, r'_i values in a signing protocol. However, all r_i, r'_i are chosen at random from $\{0,1\}^\lambda$, i.e., such an event can only occur with negligible probability. Thus, the environment can recognize this game hop only with negligible probability, i.e., the adversary cannot distinguish between the views.

Game 10: In the final game the transition from letting the challenger run the "real" protocol (w.r.t. Game 9) to letting him interact with the ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}}$ and simulate all messages based solely on the information he can obtain from $\mathcal{F}_{\mathsf{Pass2Sign}}$ is made.

Clearly, each hop changes the view of the environment only negligibly, and thus the given simulator proves that the construction is UC-secure (and correct), if all the building blocks are secure.

4.4.2 Simulator

Now, the the proof is completed by describing how to construct a simulator SIM such that for any environment $\mathcal{Z}$ and adversary $\mathcal{A}$ that controls a certain subset of the parties, the view of the environment in the real world, when running the protocol (according to GAME 9 from Section 4.4.1) with the adversary, is indistinguishable from its view in the ideal world where it interacts with the ideal functionality and the simulator (which corresponds to the final game from Section 4.4.1).

"$\mathcal{D}$","$\mathcal{S}$" denote the simulated honest party $\mathcal{D}$ and $\mathcal{S}$ respectively in the real world. The description of the simulator is given as follows: Section 4.4.2.2 describes the setup procedure for the different combinations of honest and corrupt parties. Analogously, Section 4.4.2.3 describes the signing process for those combinations. The simulation of the random oracle is then described in Section 4.4.2.4 and the handling of adaptive corruptions is given in Section 4.4.2.5. For simplicity, $\mathcal{F}_{\mathsf{Pass2Sign}}$ is now referred to as $\mathcal{F}$ from now on.

4.4.2.1 Simulation of the Initialization Protocol

When the server is initially honest, "$\mathcal{S}$" creates its public key of the non-committing encryption scheme as $\mathsf{pk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{PUBLICKEY}, 1^{\lambda})$ instead of using the real key generation, and honestly runs the code of $\mathcal{F}_{\mathsf{CA}}$. Also, if $\mathcal{S}$ has not yet registered a public key and becomes corrupted, send (INIT, sid) to $\mathcal{F}$, once the adversary decides to register a key. In other words, all calls are simply passed through.

4.4.2.2 Simulation of the Setup Protocol

Setup – Step 1 (done by honest device "$\mathcal{D}$"): The simulation starts when SIM receives a message $(\mathsf{SETUPREQ}, sid,)$ from $\mathcal{F}$, and then depends whether the account is created with an honest or corrupt server. In both cases, though, $k, (N, e), d_{\mathcal{S}}$ and $d_{\mathcal{D}}$ are generated according to the real protocol.

Server is honest. When "$\mathcal{D}$" is supposed to send an encryption of the secret key share and its authentication information, it sends a simulated ciphertext $C \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPT}, |N| + \lambda, (sid, (N, e)))$ instead. The simulator also internally stores the tuple $(C, (d_{\mathcal{S}}, \perp_{h,\mathsf{pwd}}), (sid, (N, e)))$ in a list $\mathcal{Q}$. It will determine a value for h_p as soon as the server gets corrupted. The simulator then creates an internal request record $(\mathsf{signreq\text{-}rec}, sid, k, \perp_{h,\mathsf{pwd}}, d_{\mathcal{S}}, d_{\mathcal{D}}, (N, e))$. The record will be transformed into an activated setup record when the request arrives at the server.

Server is corrupt. Here the password hash h_p is chosen at random and "$\mathcal{D}$" sends the correct ciphertext C and not a simulated one. The simulator then maintains a full setup record $(\mathsf{setup\text{-}rec}, sid, k, h_p, d_{\mathcal{S}}, d_{\mathcal{D}}, (N, e))$.

Setup – Step 2 (done by honest server "$\mathcal{S}$"): The simulation of an honest server "$\mathcal{S}$" starts when "$\mathcal{S}$" receives $(sid, (N, e), C)$ and "$\mathcal{S}$" does not have an account for sid yet. If the tuple $(sid, (N, e), C)$ was created by an honest device, the simulation continues with the first case, otherwise with the second case.

Request from honest device. If "$\mathcal{S}$" receives the same simulated ciphertext C that was sent by "$\mathcal{D}$", it does not decrypt the ciphertext but directly responds by sending sid. The simulator also stores an activated setup record as $(\mathsf{setup\text{-}rec}, sid, k, \perp_{h,\mathsf{pwd}}, d_{\mathcal{S}}, d_{\mathcal{D}}, (N, e))$, using the information from signreq-rec.

Request from corrupt device. Here the server "decrypts" the ciphertext C with label $(sid, (N, e))$ using the simulator of the non-committing encryption scheme as $(d_{\mathcal{S}}, h_p) \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{DECRYPT}, C, (sid, (N, e)))$ and stores the received information in an activated setup record as $(\mathsf{setup\text{-}rec}, sid, \perp_k, h_p, d_{\mathcal{S}}, \perp_{\mathcal{S}}, (N, e))$.

In a very special case the simulator SIM might already have a different request record $(\mathsf{signreq\text{-}rec}, sid, k', \perp_{h,\mathsf{pwd}}, d'_{\mathcal{S}}, d'_{\mathcal{D}}, (N', e'))$ for the same sid. This can happen if the request was initiated by an honest device, but never reached the honest server. If the adversary then corrupts the device, it can "reuse" the same sid from the honestly started setup request, but replace all other information. If the simulator notices such a replacement, it sends $(\mathsf{KEYGEN}, sid, 1, (N, e))$ to $\mathcal{F}$ which reflects the intrusion of the adversary and overwrites the initial password of the honest device in the ideal world by a dummy password "1".

If SIM does not have a signreq-rec record for sid yet, it sends $(\mathsf{SETUPREQ}, sid, 1)$ to $\mathcal{F}$ and subsequently inputs $(\mathsf{KEYGEN}, sid, \perp, (N, e))$ to $\mathcal{F}$.

Note that SIM uses "1" as the password of the corrupted device when creating, or overwriting the account in $\mathcal{F}$ and not the preimage of h_p, as such a preimage does not necessarily exist yet. For the further simulation this is sufficient though, as SIM only has to ensure that it invokes $\mathcal{F}$ either with the correct password (i.e., again with "1") or a wrong one (e.g., with "0").

Setup – Step 3 (done by honest device "$\mathcal{D}$"): When "$\mathcal{D}$" outputs $(\mathsf{SETUP}, sid, (N, e))$, the simulator registers the public key in the functionality by sending $(\mathsf{KEYGEN}, sid, \perp, (N, e))$ to $\mathcal{F}$, which will also deliver the message $(\mathsf{SETUP}, sid, (N, e))$ to the honest device in the ideal world.

4.4.2.3 Simulation of the Signing Protocol

Sign – Step 1 (done by honest device "$\mathcal{D}$"): When SIM receives a message $(\mathsf{SIGNREQ}, sid, qid)$ from $\mathcal{F}$, it starts the simulation of "$\mathcal{D}$" which now has to initiate a signing protocol in the real world, but without knowing m or pwd'. Again, the simulation branches depending on whether "$\mathcal{D}$" interacts with an honest or corrupt server.

Server is honest. Here "$\mathcal{D}$", instead of sending the real ciphertext C' to "$\mathcal{S}$", sends a simulated ciphertext $C' \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPT}, 3\lambda, (sid, qid))$ using the simulator of the non-committing encryption scheme. At this point, the simulator does not know the entire corresponding plaintext. Namely, it does not know h_m, t yet and also does not have enough information to "correctly" determine h'_p. Thus, SIM stores the incomplete ciphertext/plaintext tuple $(C', (\perp_{h',\mathsf{pwd}}, \perp_{h_m}, \perp_{\mathsf{tag}}), (sid, qid))$ in $\mathcal{Q}$.

The device should also have created an internal signing record. Therefore, "$\mathcal{D}$" chooses a random $r \xleftarrow{\$} \{0,1\}^\lambda$ and stores it as $(\mathsf{sign\text{-}rec}, sid, qid, r)$ and also initiates a signature request record $(\mathsf{signreq\text{-}rec}, sid, qid, \bot_{h',\mathsf{pwd}}, \bot_{h_m}, \bot_m, \bot_{\mathsf{tag}}, r, \bot_{r'}, \bot_\sigma)$ that will be used for a consistent simulation. Similar as in setup, the request record will become a "real" sign record as soon as the request arrives at the honest server.

Server is corrupt. In this case, the simulator continues by sending $(\mathsf{DELIVER}, sid, qid, \bot, \bot)$ to $\mathcal{F}$ in order to learn whether the submitted password was correct. That is, when SIM then receives $(\mathsf{SIGNREQ}, sid, qid, \mathsf{status})$ from $\mathcal{F}$ with $\mathsf{status} = \mathsf{pwdok}$ it retrieves its setup record $(\mathsf{setup\text{-}rec}, sid, k, h_p, d_\mathcal{S}, d_\mathcal{D}, (N, e))$ for sid and computes $h'_p \leftarrow \mathcal{H}(qid, h_p)$. If SIM learns that the password did not match, i.e., $\mathsf{status} = \mathsf{pwdwrong}$, the simulator chooses $h'_p \xleftarrow{\$} \{0,1\}^\lambda$ at random. Recall that SIM does not know the actual message that should be signed, but here the device has to send a correctly computed ciphertext $C' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, (h'_p, h_m, t), (sid, qid))$ to $\mathcal{S}$. Thus, in addition to r, "$\mathcal{D}$" also chooses random $h_m \xleftarrow{\$} \{0,1\}^\lambda$ and $t \xleftarrow{\$} \{0,1\}^\lambda$. All values r, h_m, h'_p, t are kept in an internal record $(\mathsf{sign\text{-}rec}, sid, qid, h'_p, h_m, \bot_m, t, r, \bot_{r'}, \bot_\sigma)$. The simulated device "$\mathcal{D}$" then sends the message (sid, qid, C') to $\mathcal{S}$.

Sign – Step 2 (done by honest server "$\mathcal{S}$"): The simulation of an honest server starts when "$\mathcal{S}$" receives a message (sid, qid, C'), and then branches depending on whether the message was sent by an honest device or not. That is, even if the device got corrupted in the meantime, but the adversary has not replaced the initially sent message of "$\mathcal{D}$", the first case applies.

Request from honest device. If (sid, qid, C') was sent by an honest device, "$\mathcal{S}$" does not decrypt C' but directly sends $(\mathsf{DELIVER}, sid, qid, \bot, \bot)$ to $\mathcal{F}$, triggering the output to $\mathcal{S}$ in the ideal world. The simulator also reflects the arrived request by storing a record $(\mathsf{sign\text{-}rec}, sid, qid, \bot_{h',\mathsf{pwd}}, \bot_{h_m}, \bot_m, \bot_{\mathsf{tag}}, r, \bot_{r'}, \bot_\sigma)$ using the information form $\mathsf{signreq\text{-}rec}$.

Request from corrupt device. When a request came from a corrupt device, SIM must further distinguish whether or not it replaces a pending sign request that was initiated from the device when it was still honest.

If it replaces another request, the simulator already maintains a $\mathsf{signreq\text{-}rec}$ record for the same qid. The simulator then first decrypt C' with the help of the simulator of the non-committing encryption scheme and obtains $(h'_p, h_m, t) \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{DECRYPT}, C', (sid, qid))$. The simulator must now reflect the replacement of the sign request towards the ideal functionality too. That is, it must determine pwd^* and m^* which should replace the initial password and message. To determine the message, the simulator uses its maintained random-oracle table $\mathcal{T}$ to look up the preimage (r, m^*) for h_m. If no preimage of the form (r, m^*) exists, SIM sets $m^* \leftarrow \bot$, which it will solely use towards the ideal functionality. Note, that all calls to $\mathcal{H}$ are also implicitly prefixed with the sid of the current session/account. That is, the adversary cannot reuse a dummy h_m that was chosen at random in another sid-session by an honest device to obtain a valid signature on the unknown message.

The simulator proceeds similarly to obtain the adversaries password attempt: SIM first checks if the random oracle table $\mathcal{T}$ contains a preimage (qid, h_p) for h'_p and a preimage

(k, pwd^*) for the retrieved h_p and with the device key k as prefix (which is stored in setup-rec for *sid*). If such a proper preimage was found, SIM uses the retrieved pwd^* or sets $\mathsf{pwd}^* \xleftarrow{\$} \{0,1\}^\lambda$ to a random value otherwise, which will mimic a signature request for a wrong password. Finally, it sends $(\mathsf{DELIVER}, sid, qid, \mathsf{pwd}^*, m^*)$ to $\mathcal{F}$ which triggers the output of $(\mathsf{SIGNREQ}, sid, qid, \mathsf{status})$ to the honest server in the ideal world. The honest server also maintains a record $(\mathsf{sign\text{-}rec}, sid, qid, h_m, t, c)$ for the signing request, where $c \leftarrow \mathsf{pwdwrong}$ if no proper pwd^* was found in the random oracle, and $c \leftarrow \perp$ otherwise (as SIM doesn't know yet whether the password was correct or wrong)

For the case where the request was fully initiated from a corrupt device, the simulator uses the same strategy as above to decrypt (h'_p, h_m, t) and determine the message m. However, the password attempt is derived differently as here the server maintains a complete setup record $(\mathsf{setup\text{-}rec}, sid, h_p, d_\mathcal{S}, (N, e))$ for *sid*. Thus, "$\mathcal{S}$" can use the stored value h_p to normally verify whether $\mathcal{H}(qid, h_p) = h'_p$. If that is the case, SIM sets $\mathsf{pwd}' \leftarrow 1$ and $\mathsf{pwd}' \leftarrow 0$ otherwise, where pwd' denotes the password that the simulator will use towards the ideal functionality (recall that SIM sets $\mathsf{pwd} \leftarrow 1$ in $\mathcal{F}$ for an account generated by the simulator). The simulator then initiates a signing session in $\mathcal{F}$ by sending $(\mathsf{SIGNREQ}, sid, qid, \mathsf{pwd}', m)$ followed by $(\mathsf{DELIVER}, sid, qid, \perp, \perp)$. Here, the honest server also maintains a full record $(\mathsf{sign\text{-}rec}, sid, qid, h_m, t, c)$ for the signing request.

In both cases, the honest server also creates activated sign records now. For values h_m where a proper preimage (r, m) in $\mathcal{T}$ existed, SIM stores a sign record $(\mathsf{sign\text{-}rec}, sid, qid, h'_p, h_m, m, t, r, \perp_{r'}, \perp_\sigma)$. That is, here SIM knows the message m that it is supposed to sign. If no preimage of the form (r, m) existed, the simulator only creates a record $(\mathsf{sign\text{-}rec}, sid, qid, h'_p, h_m, \perp_m, t, \perp_r, \perp_{r'}, \perp_\sigma)$.

Sign – Step 3 (done by honest server "$\mathcal{S}$"): When SIM receives a message $(\mathsf{PROCEED}, sid, qid)$ from $\mathcal{F}$ it knows that the password pwd' provided by the device was correct and the honest server in the ideal world approves the signing request. Thus, "$\mathcal{S}$" acts accordingly and creates its signature contribution $\sigma_\mathcal{S}$. Depending on whether "$\mathcal{S}$" has received the signing request from an honest or corrupt device, the simulator might not have chosen a value for h_m, t yet, and thus the simulation again branches:

Request from honest device. Here, "$\mathcal{S}$" only received a dummy ciphertext C' in the previous step, and the simulator has not assigned a value to h_m, t or h'_p yet. Thus, "$\mathcal{S}$" now chooses $h_m \xleftarrow{\$} \{0,1\}^\lambda$ and $r' \xleftarrow{\$} \{0,1\}^\lambda$ and uses the secret key $d_\mathcal{S}$ stored in the setup-rec record for *sid* to compute $\sigma_\mathcal{S} \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid, \mathcal{H}(r', h_m))^{d_\mathcal{S}}$. The simulator also draws $t \xleftarrow{\$} \{0,1\}^\lambda$ and updates its sign-rec record to $(\mathsf{sign\text{-}rec}, sid, qid, \perp_{h',\mathsf{pwd}}, h_m, \perp_m, t, r, r', \perp_\sigma)$.

The newly created information is also included in the list $\mathcal{Q}$ by updating the tuple with label (sid, qid) to contain the almost full plaintext $(\perp_{h,\mathsf{pwd}}, h_m, t)$. Recall that $h'_{p,i}$ is only assigned as soon as the server gets corrupted, as described in Game 7. The server now also maintains a full record $(\mathsf{sign\text{-}rec}, sid, qid, h_m, t, c)$ with $c \leftarrow \mathsf{pwdok}$ for the signing request. "$\mathcal{S}$" then sends $(sid, qid, h_m, t, r', \sigma_\mathcal{S})$ to "$\mathcal{D}$", if the device is still honest, or to $\mathcal{D}$ if it got corrupted in the meantime.

Request from corrupt device. Here, "$\mathcal{S}$" looks up its record (sign-rec, sid, qid, h'_p, h_m, $\{m, \perp_m\}$, t, $\{r, \perp_r\}$, $\perp_{r'}$, $\perp_\sigma$) and normally computes its signature share $\sigma_\mathcal{S}$ for the stored h_m. In that process, "$\mathcal{S}$" chooses a random $r' \xleftarrow{\$} \{0,1\}^\lambda$ which is then included in the sign-rec record. Finally, "$\mathcal{S}$" sends $(sid, qid, h_m, t, r', \sigma_\mathcal{S})$ to $\mathcal{D}$. The honest server also updates its state record to (sign-rec, sid, qid, h_m, t, c) setting $c \leftarrow$ pwdok.

Here, SIM now maintains a signature record of the form (sign-rec, sid, qid, h'_p, h_m, $\{m, \perp_m\}$, t, $\{r, \perp_r\}$, r', $\perp_\sigma$) That is, SIM knows (for "well-formed" h_m) the message m it has provided its signature share for, but not the created signature σ. In particular, also the ideal functionality $\mathcal{F}$ has not stored any signature for m yet. However, SIM can provide the missing signature right on time when the environment tries to verify the signature, as described in Section 4.4.2.4. Note, for non "well-formed" hashes h_m SIM has initiated a signature request for dummy message $m = \perp$ towards the ideal functionality. However, such a request will never lead to a signature in $\mathcal{F}$ due to the collisision-resistance provided by the random oracle.

Sign – Step 4 (done by honest device "$\mathcal{D}$"): When "$\mathcal{D}$" receives a message $(sid, qid, h_m, t, r', \sigma_\mathcal{S})$, it only continues if the received value t is the same as stored in (sign-rec, sid, qid, h'_p, h_m, $\perp_m$, t, r, r', $\perp_\sigma$). It then completes the signature to σ_{RSA} using the locally stored $d_\mathcal{D}$ value. Eventually, "$\mathcal{D}$" ends with output (SIGNATURE, sid, qid, σ), upon which SIM sends (SIGNATURE, sid, qid, σ) to $\mathcal{F}$ and also stores σ_{RSA} in its internal sign-rec record. Thus, the simulator has then "blindly" generated a signature in $\mathcal{F}$, i.e., SIM has not learned the signed message m yet, while the ideal functionality now contains a record (signature-rec, (N, e), m, σ, true).

To summarize, depending on the interference of the adversary and the input of the environment SIM ends this simulation for an *honest* device in one of the following states:

Signature completed: The signature process ended correctly and $\mathcal{F}$ contains a valid signature record including the message m (which is unknown to the simulator) and the completed signature σ. The simulator created a signature record (sign-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, h_m, t, $\perp_m$, r, r', σ_{RSA}), that contains h_m and the created RSA signature σ_{RSA}, and where $\perp_m$ stands for the unknown message m that the simulator had signed. However, the simulator can use the fact that the ideal functionality contains a completed signature record *including* the message m to ensure consistency when a random oracle query for the message (r, m) is made. Furthermore, if done with an honest server "$\mathcal{S}$", "$\mathcal{S}$" also created a signing record (sign-rec, sid, qid, h_m, t, c) where $c \leftarrow$ pwdok.

Proceed, but no completed signature: If SIM ended in the second state when the environment gave the ok to proceed, but the adversary in the real world has interrupted the final message, such the device never received $(sid, qid, h_m, t, r', \sigma_\mathcal{S})$ from "$\mathcal{S}$". Consequently, SIM could not create the signature record in $\mathcal{F}$ yet, that SIM later needs to ensure consistency with the random oracle. Also the simulator only maintains a record (sign-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, h_m, t, $\perp_m$, r, r', $\perp_\sigma$). (Note that r', t and h_m were added to the record already when "$\mathcal{S}$" send its message.) However, SIM will finalize the computation of σ as soon as the device gets corrupted using the already selected r' and create the

corresponding ideal world record by sending (SIGNATURE, sid, qid, σ) to $\mathcal{F}$. If an honest server was involved, it also stores a full record (sign-rec, sid, qid, h_m, t, c) with $c \leftarrow$ pwdok.

No proceed: The third state occurs when the sign request arrived at the server, but the either the environment didn't gave the ok to proceed. Thus, SIM has neither created a signature $\sigma_{\mathcal{S}}$, nor allocated r', t or h_m yet. The record of SIM therefore looks as follows (sign-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, $\perp_{h_m}$, $\perp_m$, $\perp_{\mathsf{tag}}$, r, $\perp_{r'}$, $\perp_\sigma$). An honest server is then also supposed to have created an intermediate record (sign-rec, sid, qid, h_m, t, c) where c indicates whether the password matched or not. However, in the simulation h_m and c are not known yet, and therefore "$\mathcal{S}$" only has the incomplete record (sign-rec, sid, qid, $\perp_{h_m}$, $\perp_{\mathsf{tag}}$, $\perp_c$). The simulator will complete such records when the server gets corrupted.

Signing request never arrived: The last state occurs when the adversary already intercepted the signature request from "$\mathcal{D}$". Here, SIM only holds the following record (signreq-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, $\perp_{h_m}$, $\perp_m$, $\perp_{\mathsf{tag}}$, r, $\perp_{r'}$, $\perp_\sigma$).

4.4.2.4 Verification and Random Oracle Simulation

In the simulation so far, SIM has sometimes "blindly" signed messages for a user, or signed messages which were known to the simulator, but where SIM did not learn the resulting signature. However, SIM can use the verification interface and the fact that SIM is in charge of answering the random oracle queries, to learn the missing values and ensure consistency with the values maintained by the ideal functionality $\mathcal{F}$.

First is shown how the simulator SIM learns the missing message for records of the type (sign-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, h_m, $\perp_m$, t, r, r', σ_{RSA}) or records of the form (signreq-rec, sid, qid, $\{h'_p, \perp_{h',\mathsf{pwd}}\}$, h_m, $\perp_m$, t, r, r', σ_{RSA}). The simulator uses the procedure as described in Game 8 and Game 9, with the modification that whenever the challenger would look up if a record (signature-rec, sid, qid, m, σ) exists, SIM sends (VERIFY, $sid, m, \sigma, (N, e)$) to $\mathcal{F}$, where (N, e) is taken from the setup record for sid. When $\mathcal{F}$ asks SIM to verify a signature σ, the simulator returns (VERIFIED, $sid, m, \sigma, (N, e)$, false) and waits for a message (VERIFIED, $sid, m, \sigma, (N, e), f$). If $f =$ true, SIM behaves as if the challenger would have found a matching record (signature-rec, sid, qid, m, σ). Thereby SIM ensures that whenever a random oracle query (r, m) is made where r was used in a signing session, it can detect a query containing the "real" m that the simulator might have blindly signed, and react consistently.

The second type of incomplete records were created when only the server was honest. In that case the simulator could "extract" the message, but did not learn the signature the corrupt device had completed. Thus, the simulator maintains a record (sign-rec, sid, qid, h'_p, h_m, m, t, r, r', $\perp_\sigma$), and more importantly, the ideal functionality did not receive the signature either. Whenever SIM then receives a message (VERIFY, $sid, m, \sigma, (N, e), \mathcal{P}$) from $\mathcal{F}$, it checks whether σ is valid signature on m using the normal verification algorithm. If verification fails, SIM responds with (VERIFIED, $sid, m, \sigma, (N, e)$, false) to $\mathcal{F}$. When the verification succeeds, SIM parses σ as (σ_{RSA}, qid, r, r') and checks if it has a setup-rec record for sid with the same public key (N, e) and also a record (sign-rec, sid, qid, h'_p, h_m, m, t, r, r', $\perp_\sigma$) for the same sid, qid, m, r, r'. If such matching records are found, i.e., the signature belongs to a previous signing request, SIM sends (SIGNATURE, sid, qid, σ) to $\mathcal{F}$, ensuring that the signature is now registered by the ideal functionality as well, followed by a call (VERIFIED, $sid, m, \sigma, (N, e)$, true). When the verification

in the real world succeeded but no matching record was found, SIM aborts, as justified in GAME 5 and GAME 6.

The simulator also takes special care of queries that contain the device secret key k as described in GAME 8 and GAME 9. Whenever, in those games a record signreq-rec or setupreq-rec is used, SIM instead invokes the PWDGUESS interface of $\mathcal{F}$. This procedure is recalled in more detail in the description of full corruption below.

4.4.2.5 Adaptive Corruptions

Device Corruption. When the environment decides to corrupt a previously honest device, "$\mathcal{D}$" receives the message (CORRUPT, sid) from the environment. This is mimicked by sending (CORRUPT, $sid, \mathcal{D}, \emptyset$) to $\mathcal{F}$ in the ideal world. The impact of that corruption depends on whether it happened after the setup was completed, or before, and whether or not the server is honest.

During setup: If the device gets corrupted during setup, i.e., "$\mathcal{D}$" already had sent (sid, $(N, e), C$) to the server but never completed the setup, it must provide its setup record (setup-rec-temp, $sid, k, d_{\mathcal{D}}, (N, e)$) to the adversary. All values were correctly generated in the setup and thus, "$\mathcal{D}$" simply gives the record (setup-rec-temp, $sid, k, d_{\mathcal{D}}, (N, e)$) to $\mathcal{A}$. If the setup was done with an honest server which already replied with (sid), but the adversary never let the message arrive at "$\mathcal{D}$", the simulator now also sends (SETUP, sid, (N, e)) to $\mathcal{F}$. This ensures that the account will be activated in the ideal functionality as well.

After setup: When the device gets corrupted after the setup was done, the adversary expects to get the setup record (setup-rec, $sid, k, d_{\mathcal{D}}, (N, e)$) as well as all records $\{$(sign-rec, qid_i, r_i)$\}$ that result from signing requests.

Before the simulator outputs those records, it ensures that incomplete signature requests that were initiated by the device are now completed towards the ideal functionality. This is crucial to ensure consistency with a potential random oracle query (r_i, m_i) as described in Section 4.4.2.4. The type of discontinued signing request SIM can complete now depends on whether the server is (still) honest or not.

- *Server honest:* If the server is still honest, the simulator can only complete signing requests, where the honest server (in the ideal world) already gave the ok to proceed. That is, for those requests, where in the simulation "$\mathcal{S}$" had already send its contribution $(r'_i, \sigma_{\mathcal{S},i})$ to "$\mathcal{D}$" but the adversary intercepted the share. For each such intercepted signing session the simulator maintains a record (sign-rec, sid, $qid_i, h'_p, h_{m,i}, \perp_m, t_i, r_i, r'_i, \perp_\sigma$) and now computes $(\sigma_{\mathsf{RSA},i}) \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m,i}))^{d_{\mathcal{S}}\, d_{\mathcal{D}}}$ using the RSA secret key shares $d_{\mathcal{S}}, d_{\mathcal{D}}$ the honest device had generated in setup. It then sets $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$, updates its record sign-rec to contain $\sigma_{\mathsf{RSA},i}$ and, most importantly, sends (SIGNATURE, sid, qid_i, σ_i) to $\mathcal{F}$.
  From now on the simulator must also answer to random oracle queries (r_i, m_i) where r_i belongs to a signing session where the honest server had never sent its signature share. In those cases the adversary has not learned $h_{m,i}$ though, and in fact, those are not even chosen by the simulator yet. Thus, SIM will respond with a random value $h_{m,i}$ for each

such query. The simulator then has to ensure consistency for those queries as soon as the server gets corrupted as well.

- *Server corrupt:* If the server was initially honest, the simulator first completes the records as in the case of the honest server described above. However, given that here the server is corrupt, the simulator additionally creates signatures in $\mathcal{F}$ for those sessions, where i) either an initially honest server never got the approval to continue or ii) a corrupt server did not provide a (valid) signature share.For that type of discontinued signing requests, the simulator maintains records $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \perp_m, t_i, r_i, \perp_{r'}, \perp_\sigma)$. In case those records stem from a setting where the server was initially honest, but got then corrupted, $h'_{p,i}, h_{m,i}, t_i$ got assigned at the moment of the corruption (see description of server corruption).

  The simulator SIM now completes each such $\mathsf{sign\text{-}rec}$ record by choosing $r'_i \xleftarrow{\$} \{0,1\}^\lambda$ and computing $\sigma_{\mathsf{RSA},i} \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid_i, \mathcal{H}(r'_i, h_{m,i}))^{d_\mathcal{S} d_\mathcal{D}}$, again using the knowledge of $d_\mathcal{S}, d_\mathcal{D}$, generated by the honest device. SIM then sends $(\mathsf{PROCEED}, sid, qid_i)$ to $\mathcal{F}$, followed by the message $(\mathsf{SIGNATURE}, sid, qid_i, \sigma_i)$ where the signature is composed as $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$.

In both cases the simulator has now created full signature records in $\mathcal{F}$ that maps the still unknown message m_i to the "blindly" created signature σ_i. This will allow the simulator to determine m_i whenever a query (r_i, m_i) is made to the random oracle from now on (which is exactly the crucial turning point, as the adversary will learn all the r_i values of the incomplete signing records, which thus need to be simulated), using the procedure described in Section 4.4.2.4.

Server Corruption. When the honest server "$\mathcal{S}$" receives the message $(\mathsf{CORRUPT}, sid)$ from the adversary, the adversary then expects to learn the secret key of the non-committing encryption scheme $\mathsf{sk_{ENC}}$, the setup information $(\mathsf{setup\text{-}rec}, sid, h_p, d_\mathcal{S}, (N, e))$ and records $\{(\mathsf{sign\text{-}rec}, sid, qid_i, h_{m,i}, t_i, c_i)\}$ for all signing sessions.

However, h_p might still be unassigned and the signing records will currently have the form $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h_m}, \perp_{\mathsf{tag}}, \perp_c)$ though, whenever they were created in a signing protocol with an honest device and the server did not continue the protocol. However, SIM can assemble the correct records now using the list $\mathcal{L}$ that SIM will obtain from $\mathcal{F}$ and the internal signing records maintained by SIM. How the simulation proceeds again depends on whether the device is still honest or not.

- *Device honest:* If the device is still honest, SIM sends $(\mathsf{CORRUPT}, sid, \mathcal{S}, \emptyset)$ to $\mathcal{F}$ receiving $(\mathsf{CORRUPT}, sid, \mathcal{S}, \mathcal{L})$. Then, for every record $(\mathsf{sign\text{-}rec}, sid, qid_i, \perp_{h_m}, \perp_{\mathsf{tag}}, \perp_c)$ stored by "$\mathcal{S}$", the simulator takes c_i from the tuple $(qid_i, c_i) \in \mathcal{L}$ and includes it in the $\mathsf{sign\text{-}rec}$ record. Determining the missing values for h_p, $h'_{p,i}$, $h_{m,i}$ and t_i is also rather simple here, as the adversary has not learned the device key k and randomness r_i yet that was used to "blind" the message m_i in $h_{m,i}$. Thus, SIM also had not to react to random oracle queries of the form (r_i, m_i) yet. Thus, for each incomplete $\mathsf{sign\text{-}rec}$ record the simulator simply chooses random values $h_{m,i} \xleftarrow{\$} \{0,1\}^\lambda$, $t_i \xleftarrow{\$} \{0,1\}^\lambda$. The password hash h_p is chosen at random now and whenever $c_i = \mathsf{pwdok}$ it also sets the password attempt of that session to $h'_{p,i} \leftarrow \mathcal{H}(qid, h_p)$

and to $h'_{p,i} \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. The created values of h_p, $h'_{p,i}$, $h_{m,i}$ are then added to the setup-rec and sign-rec records.

- *Device corrupted during setup:* When the adversary corrupted the initially honest device during setup it has learned the device secret k. Thus, the adversary would already have been able to make a random oracle query to compute the password hash himself that is supposedly encrypted in C. The simulator has to figure out whether he already committed to h_p before outputting the secret decryption key $\mathsf{sk}_{\mathsf{ENC}}$.
  To do so, SIM sends $(\mathsf{CORRUPT}, sid, \mathcal{S}, \emptyset)$ to $\mathcal{F}$ receiving $(\mathsf{CORRUPT}, sid, \mathcal{S}, \mathcal{L})$ which will also enable the password guess interface. The simulator then goes through previously answered random oracles queries that had the form $\mathcal{H}(k, \mathsf{pwd}_j)$, and for each sends $(\mathsf{PWDGUESS}, sid, \bot, \mathsf{pwd}_j)$ to $\mathcal{F}$ to verify whether pwd was the actual password of the initially honest device. If $\mathcal{F}$ responds with $(\mathsf{PWDGUESS}, sid, \bot, c_j)$ with $c_j = \mathsf{pwdok}$, it sets h_p to be the random oracle answer it had randomly assigned for that query. If no such matching query is found, h_p is chosen at random. To determine the decryption key, simulator invokes $\mathsf{sk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$ with $\mathcal{Q} \leftarrow (C, (d_{\mathcal{S}}, h_p), (sid, (N, e))$ using (N, e), $d_{\mathcal{S}}$ from the signreq-rec record. Finally, "$\mathcal{S}$" outputs $\mathsf{sk}_{\mathsf{ENC}}$ as well as all sign records. Note that here all sign records where already completed during the protocol run as all requests originated from a corrupt device, i.e., nothing has to be simulated here.
- *Device corrupted after setup:* When the device is corrupted sometime after setup, SIM has to take special care of all signing request that were initiated when the device was still honest. Here the adversary when corrupting the device has not only learned the device secret k but also all records $\{(\mathsf{sign\text{-}rec}, qid_i, r_i)\}$ of signing requests from "$\mathcal{D}$". Thus, the adversary could have made random oracle queries targeted to compute the message hash himself. More precisely, the adversary would have been able to query (r_i, m_i) to the random oracle where m_i is the message for which the environment triggered the signing request. As the adversary will now also learn the decryption key $\mathsf{sk}_{\mathsf{ENC}}$, SIM has to make sure that all ciphertexts C'_i sent by "$\mathcal{D}$" in the signing requests now open to the correct plaintext. That is, it must hold that $(h'_{p,i}, h_{m,i}, t_i) \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, C'_i, (sid, qid_i))$ and $h_{m,i} \leftarrow \mathcal{H}(r_i, m_i)$.
  This requires a more careful simulation using the functionality to determine the blindly signed messages. The simulator now completes the signature record in $\mathcal{F}$ and uses that record to determine whether it already answered to a random oracle query (r_i, m_i). To this end, SIM first retrieves each qid_i for which a request record $(\mathsf{signreq\text{-}rec}, sid, qid_i, \bot_{h',\mathsf{pwd}}, \bot_{h_m}, \bot_m, \bot_{\mathsf{tag}}, r_i, \bot_{r'}, \bot_\sigma)$ exists and there is either a matching sign record of the form $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, *, *, *, *, *)$ or $(\mathsf{sign\text{-}rec}, sid, qid_i, \bot_{h',\mathsf{pwd}}, \bot_{h_m}, *, *, *, \bot_{r'}, \bot_\sigma)$. For each such qid_i, SIM chooses $r'_i \xleftarrow{\$} \{0,1\}^\lambda$, a *random* $h'_{m,i} \xleftarrow{\$} \{0,1\}^\lambda$ and computes the full signature $\sigma_{\mathsf{RSA},i} \leftarrow \mathcal{H}_{\mathsf{RSA}}(sid, qid_i, h'_{m,i})^{d_{\mathcal{S}} d_{\mathcal{D}}}$, again using the knowledge of $d_{\mathcal{S}}, d_{\mathcal{D}}$, generated by the initially honest device. SIM then updates it (empty) sign-rec record to contain $\sigma_{\mathsf{RSA},i}$ and r'_i. The simulator also adds σ_i with $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$ to a signature list Σ. If the list is complete, SIM finally sends $(\mathsf{CORRUPT}, sid, \mathcal{P}, \Sigma)$ to $\mathcal{F}$ which will generate full signature records in $\mathcal{F}$ for all blindly signed message m_i that were incomplete so far. The input also enables the PWDGUESS interface, which will be crucial for the rest of the simulation.
  SIM first leverages the fact that now all blindly signed messages are completed within $\mathcal{F}$ to ensure a consistent decryption for the dummy ciphertexts C'_i of the sign protocols. The following procedure is done to either complete *request* records or sign records. In the former

case, the initial honest sign request was replaced after device corruption, whereas in the latter one the sign request was received by the server but never completed.
Now, SIM goes through all random oracle queries of the form $((r_i, m_i), h_{m,i}) \in \mathcal{T}$ for which a signreq-rec record for r_i exist and sends $(\mathsf{VERIFY}, sid, m_i, \sigma_i, (N, e), \mathsf{SIM})$ to $\mathcal{F}$, where again $\sigma_i \leftarrow (\sigma_{\mathsf{RSA},i}, qid_i, r_i, r'_i)$ and (N, e) is taken from the setup record for the sid specified in sign-rec. The ideal functionality will then send its ping $(\mathsf{VERIFY}, sid, m_i, \sigma_i, (N, e), \mathsf{SIM})$ to SIM, upon which it responds with $(\mathsf{VERIFIED}, sid, m_i, \sigma_i, (N, e), \mathsf{false})$. In case m_i is indeed the blindly signed message of signing request qid_i, the ideal functionality will respond with $(\mathsf{VERIFIED}, sid, m_i, \sigma_i, (N, e), \mathsf{true})$. Whenever that happens, SIM now knows that it had signed a message m_i and also assigned already a hash value $h_{m,i} \leftarrow \mathcal{H}(r_i, m_i)$ for m_i. Thus, SIM includes that hash value $h_{m,i}$ in its internal record sign-rec or signreq-rec. It also "links" the signed $h'_{m,i}$ value to $h_{m,i}$ by setting $\mathcal{H}(r'_i, h_{m,i}) \leftarrow h'_{m,i}$. If for a signing query qid_i no such matching random oracle query (r_i, m_i) was found, SIM draws a random hash $h_{m,i} \xleftarrow{\$} \{0,1\}^\lambda$ and updates its internal sign-rec or signreq-rec record accordingly. In addition, it adds valid tags $t_i \leftarrow \mathcal{H}(\texttt{"MAC"}, qid, k, h_{m,i})$ to the records using k from the corresponding setup-rec record. SIM also has to take similar care of the password hashes, as the adversary already knows k and would have been able to compute the password hashes himself. Thus SIM uses the procedure from Game 7 to determine h_p and all $h'_{p,i}$ such that they are consistent with the adversaries view. However, in difference to Game 7 SIM does not have internal records for the password pwd and all password attempts pwd'_i. Instead, SIM uses the PWDGUESS interface of $\mathcal{F}$, which is available now since both parties are corrupt. Thus, for all previously answered random oracles queries that had the form $\mathcal{H}(k, \mathsf{pwd})$, SIM sends $(\mathsf{PWDGUESS}, sid, \bot, \mathsf{pwd})$ to $\mathcal{F}$ to verify whether pwd was the actual password of the initially honest device. If such a query is found, it sets h_p to be the random oracle answer it had randomly assigned for that query. Likewise, for all queries $\mathcal{H}(qid_i, h^*_p)$ where $((k, \mathsf{pwd}'_i), h^*_p) \in \mathcal{T}$, SIM send $(\mathsf{PWDGUESS}, sid, qid_i, \mathsf{pwd}'_i)$ to $\mathcal{F}$ and reuses its previous random oracle answers for each $h'_{p,i}$ it had already created.
Note that all internal records will now have the form $(\mathsf{sign\text{-}rec}, sid, qid_i, h'_{p,i}, h_{m,i}, \bot_m, t_i, r_i, r'_i, \sigma_{\mathsf{RSA},i})$ and a corresponding full signature record in $\mathcal{F}$ was created. That is, for all signature request that were ever started by the honest device, the simulator can now use $\mathcal{F}$ to ensure consistency with random oracle queries (r_i, m_i) of still unknown messages m_i (as described in Section 4.4.2.4).

In the first and third cases, SIM now uses its sign-rec records to assemble the complete signing records $\{(\mathsf{sign\text{-}rec}, sid, qid_i, h_{m,i}, t_i, c_i)\}$ the server is supposed to give to $\mathcal{A}$. Further, the simulator adds the newly obtained values to the list $\mathcal{Q}$ to contain the full plaintext tuples $(h'_{p,i}, h_{m,i}, t_i)$ for every simulated ciphertext C'_i. It also adds such tuples for all replaces signature request, i.e., where the simulator just completed the signreq-rec records. Similarly, it adds $C, (d_{\mathcal{S}}, h_p), (sid, (N, e)$ to $\mathcal{Q}$ to ensure consistency for the dummy ciphertext of the setup protocol. Finally, the simulator invokes $\mathsf{sk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$ to learn the secret key of the encryption scheme. "$\mathcal{S}$" then outputs $\mathsf{sk}_{\mathsf{ENC}}, (\mathsf{setup\text{-}rec}, sid, h_p, d_{\mathcal{S}}, (N, e))$ and the completed records $\{(\mathsf{sign\text{-}rec}, sid, qid_i, h_{m,i}, c_i)\}$ of all signing sessions.

Simulating Offline Attacks after Full Corruption. When the environment has fully corrupted both parties, the adversary in the real world learned $k, h_p, h'_{p,1}, \ldots, h'_{p,q}$ where q

denotes the number of signing requests initiated by the device. Thus, it can run offline attacks against the password hashes, trying to determine the underlying password. In fact, as the passwords of honest devices were provided by the environment, they might even be known to the adversary. SIM uses $\mathcal{F}$'s PWDGUESS interface that is available for fully-corrupted instances from now on for each random oracle query that looks like an attempt of the adversary to verify the password against a learned hash value.

That is, for each query $\mathcal{H}(k, \mathsf{pwd}^*)$ where k appears in a record (signreq-rec, $sid, k, h_p, d_\mathcal{S}$, $d_\mathcal{D}, (N, e)$), SIM sends (PWDGUESS, $sid, \perp, \mathsf{pwd}^*$) to $\mathcal{F}$. When it receives (PWDGUESS, sid, qid, c) with $c = \mathsf{pwdok}$ from $\mathcal{F}$, the simulator sets $\mathcal{H}(k, \mathsf{pwd}^*) \leftarrow h_p$ where h_p is taken from the setup-rec record and to a random value when $c = \mathsf{pwdwrong}$.

For every query $\mathcal{H}(qid_i, h_p^*)$ where $h_p^* = \mathcal{H}(k, \mathsf{pwd}^*)$, i.e., h_p^* is the result of a previous random oracle query (k, pwd^*) and records (setup-rec, $sid, k, h_p, d_\mathcal{S}, d_\mathcal{D}, (N, e)$) for k and (signreq-rec, sid, $qid_i, h'_{p,i}, h_{m,i}, \{m_i/\perp_m\}, t_i, r_i, r'_i, \{\sigma_{\mathsf{RSA},i}, \perp_\sigma\}$) for qid_i exist, SIM sends (PWDGUESS, sid, qid_i, pwd^*) to $\mathcal{F}$. When $\mathcal{F}$ responds with the message (PWDGUESS, sid, qid_i, c) where $c = \mathsf{pwdok}$, SIM sets $\mathcal{H}(qid_i, h_p^*) \leftarrow h'_{p,i}$ where $h'_{p,i}$ is taken from the sign-rec record and assigns a random response otherwise.

All other parties, which only verify signatures, can be simulated honestly w.r.t. to the execution history, ignoring the programming, which has been explained in detail earlier this section.

Thus, it was shown how to construct a simulator that provides a view that is indistinguishable to the one described in Game 10, which concludes the proof.

4.5 Implementation

This section contains a short summary of a prototypical implementation of the Pass2Sign scheme. A more detailed description is given in Appendix G. The measurements of the protocol were done with three different RSA-moduli sizes, 1,024, 2,048 and 4,096Bit to account for different security requirements. The key size is used for both the signing key and the trapdoor permutation in the non-committing encryption scheme.

To instantiate the random oracles $\mathcal{K}, \mathcal{G}$, and $\mathcal{H}$ are instantiated as SHA-512, while each call is prefixed accordingly. The instantiation of the full-domain hash $\mathcal{H}_{\mathsf{RSA}}$ is based on the construction given by Bellare and Rogaway [BR93], and uses rejection sampling to uniformly map into $\mathbb{Z}_N^*$. All messages are sent using standard TCP-Sockets.

The implementation was done using Java 8 without any optimization. The server is a laptop with a 2.7GHz processor and 16GiB RAM, while the device is a Nexus 10 tablet with 1.7GHz, 2GB RAM and Android 5.1.1, while $\mathcal{F}_{\mathsf{Auth}}$ was implemented using standard TLS connections with the corresponding certificates.

Table 4.1 depicts the average time for the setup and signing protocol, split between the device and server part, based on measurements of 100 protocol runs. The table does not include network latencies, as they strongly depend on the actual location setting. However, assuming a round-trip time takes 100ms, a full signing protocol with 2,048Bit keys then requires roughly 250ms in total.

Summarized, the protocol can be considered practical, even though no optimizations were implemented.

Table 4.1: Overview of the measurements. All values are in milliseconds.

	Setup			Signing		
Key Size	1,024Bit	2,048Bit	4,096Bit	1,024Bit	2,048Bit	4,096Bit
Device						
Median	648.11	3′335.34	14′343.46	19.08	79.83	482.60
Average	855.58	3′646.27	16′202.58	19.79	83.40	574.41
Server						
Median	14.32	63.96	388.11	11.76	64.53	456.38
Average	15.20	65.69	393.27	12.31	65.50	466.73

4.6 Non-Blind Signatures

The Pass2Sign scheme guarantees message blindness towards the server, meaning that the server does not learn the message the device wishes to be signed. This may not always be required or wanted though, e.g., if the message is public or jointly determined or if the server should have control over the messages being signed (for instance because it should also apply throttling based on the message). Sketched in this section is a variant Pass2Sign* of the scheme that does not include message blindness, and comes with the additional benefit that the signing protocol is even simpler and verification requires less message pre-processing. Thus, the sketched alterations lead to an additional efficiency gain, but offer less privacy. Clearly, it depends on the use-case which protocol should be deployed in what context.

The Ideal Functionality. The ideal functionality $\mathcal{F}_{\mathsf{Pass2Sign}^*}$ can be obtained from the one for Pass2Sign by simply including the message m in the output to the server. That is, when the server learns about a signature request, the output is augmented to contain the message m provided by the device: in the `<5.Sign Delivery>` interface, the server $\mathcal{S}$ now receives $(\mathsf{SIGNREQ}, sid, qid, \mathsf{status}, m)$.

Note that so far the message still remains confidential between $\mathcal{D}$ and $\mathcal{S}$, disregarding the length the message. If the message is supposed to be entirely public, then m must also be included in the output $(\mathsf{SIGNREQ}, sid, qid, \mathcal{D}, m)$ to the adversary $\mathcal{A}$ in the `<4.Sign Request>` interface.

The Protocol. Now is sketched how the Pass2Sign realization can be modified to one that realizes $\mathcal{F}_{\mathsf{Pass2Sign}^*}$ with entirely public messages. First note that only the signing protocol needs to be changed. In a nutshell, one simply drops all its steps that aim at providing message blindness, such as the hashing of the message including r and r'. More precisely, when requesting a signature, the device drops the blinding Step 1b) where $h_m \leftarrow \mathcal{H}(r, m)$ is computed for a random r, and the "MAC" Step 1d) which aims at ensuring message consistency without having to store m on the device. Instead, the device now simply keeps the message in its sign record. Depending on whether one aims at the confidential or public message setting, the message is sent either encrypted or in plain to $\mathcal{S}$. For the latter, one has to ensure that the adversary cannot tamper with the message during delivery, and thus one would have to include m in the

label of the ciphertext. That is, for the public message setting, the device sends to $\mathcal{S}$ the tuple (sid, qid, C', m) with C' becoming $C' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, h'_p, (sid, qid, m))$.

The changes to the server's computation are similar: in Step 3b) the server drops its randomness r' and double-hashing contribution and simply signs the (slightly augmented) message (sid, qid, m) based on the received message m. The server then returns $(sid, qid, \sigma_{\mathcal{S}})$ to the device.

Finally, in Step 4c), where the device completes the signature to σ_{RSA}, the verification of σ_{RSA} is adapted accordingly and, in Step 4d), the full signature is set to $\sigma = (\sigma_{\mathsf{RSA}}, qid)$. Verification of a signature σ on message m is simplified to a standard RSA-FDH verification of σ_{RSA} for message $m' \leftarrow (sid, qid, m)$.

It is easy to see that the proof of the simplified non-blind scheme $\mathsf{Pass2Sign}^*$ can be derived with minor modifications from the proof of the $\mathsf{Pass2Sign}$ scheme. Roughly, one has to adapt the games for the reduction to the RSA assumption, and drop all simulation that stems from the message blinding, such as the simulation of the tags t or the "late-programming" of the random oracle upon a signature verification request (as the messages are now known to the simulator).

Clearly, the proofs carry over with only minor adjustments.

4.7 Conclusion

This chapter introduced a protocol for signing messages with the help of a device and a server. To authenticate towards the server, the user has to enter a password on his device. If the device gets stolen, an adversary is limited to online password guessing attacks, which can be throttled by the server. Neither the device nor the server are required to be tamper-resistant in any form, yet the protocol offers comparable security to trusted hardware, but without its inconveniences. The UC-formulation guarantees that the protocol remains secure even in arbitrarily chosen contexts. Moreover, the protocol is secure against adaptive corruptions, which properly models the main threat where the device gets lost or stolen. The model of corruption is even stronger than the existing standard: the simulator does not learn any previous inputs. The ideal functionality also provides a realistic way how password guesses are handled. Namely, even if both the device and the server are corrupted, the adversary does not immediately learn the passwords, but can only mount an offline attack. Thus, if strong passwords are used, the adversary might still not be able to guess them, despite having corrupted *both* entities. The protocol is round-optimal and very efficient, as it only requires few random oracle calls and three full-size modular exponentiations for each signature generation. Furthermore, it is sketched how to lift the blindness property from the functionality and protocol, yielding an even more efficient scheme. Possible extensions include achieving full blindness with unlinkability of the resulting signature, an instantiation in the standard model, but also extensions to more than one server.

Chapter 5

UC-Secure Non-Interactive Public-Key Encryption

The results of this chapter have already been published [CLNS17].

Abstract. The universal composability (UC) framework enables the modular design of cryptographic protocols by allowing arbitrary compositions of lower-level building blocks. Public-key encryption is unarguably a very important such building block. However, so far no UC-functionality exists that offers non-interactive encryption necessary for modular protocol construction. This chapter provides an ideal functionality for non-committing encryption (i.e., public-key encryption secure against adaptive corruptions) with locally generated, and therefore non-interactive, ciphertexts. As a sanity check, a game-based security notion is provided, which is proven to be equivalent to the UC notion. It is then shown that the encryption scheme given in the prior chapter, based on trapdoor permutations, securely implements the new notion in the random-oracle model even without assuming secure erasures. This is the best one can hope to achieve as standard-model constructions do not exist due to the uninstantiability of round-optimal non-interactive adaptively secure message transfer in the standard model [Nie02]. The modular reusability of the functionality is illustrated by constructing the first non-interactive signcryption scheme secure against adaptive corruptions without secure erasures in the UC framework.

Roadmap. The problem statement is given in Section 5.1, while Section 5.1.3 contains additional preliminaries. The new game-based definition is given in Section 5.2, which is compared to existing notions in Section 5.3. The corresponding UC-definition $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, along with a proof of equivalence, is given in Section 5.4. How to build UC-secure secure message transfer from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is presented in Section 5.5, while the primitive of UC-secure sugncryption, i.e., $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, is given in Section 5.6. The chapter is concluded in Section 5.7.

5.1 Introduction

The universal composability (UC) framework [Can01] enables the modular design of cryptographic protocols by analyzing the security of composed protocols in a so-called "hybrid" model,

where subprotocols are replaced by their ideal functionalities, thereby eliminating the need for explicit reductions from the security of the individual building blocks in the overall security proof. Faithful to its name, the UC framework guarantees that any secure instantiation of the subprotocols yields a secure instantiation of the composed protocol.

A wide variety of UC-secure cryptographic primitives have appeared in the literature. Public-key encryption is an important such primitive that may not have received the attention that it deserves in the UC framework, especially given its widespread use to build cryptographic protocols. The original UC paper by Canetti [Can01] specifies an ideal functionality $\mathcal{F}_{\mathsf{PKE}}$ for public-key encryption and proves it equivalent to the game-based notion of indistinguishability against chosen-ciphertext attack (IND-CCA2). However, this result only holds for non-adaptive adversaries, that is, where the set of corrupted parties is fixed a priori for each protocol instance.

Non-Committing Encryption. Security against adaptive corruptions is obviously the more realistic notion for practical applications, but is notoriously difficult to achieve for public-key encryption because of the so-called *selective de-commitment problem* [DNRS99, Hof11]. In a nutshell, the problem is the following. As long as both sender and receiver are honest, the simulator communicates "dummy" ciphertext values without knowing the actual plaintext. When later the receiver gets corrupted, the simulator must provide the adversary with a decryption key that makes these dummy ciphertexts decrypt to the correct messages, which are only handed to the simulator at the moment of corruption. Similarly, when the sender is corrupted, the simulator must provide encryption randomness that turns the messages into the dummy ciphertexts. One can easily solve the latter problem by securely erasing the randomness after encryption. However, secure erasures do not help to provide correct decryption keys when the receiver gets corrupted, and security cannot be proven. This is an unsatisfactory situation, as already noted in prior work [BHK12, FHKW10, FHKP16, HRW16, HR14].

Adaptively secure public-key encryption is therefore often referred to as *non-committing encryption* (NCE). Nielsen [Nie02] showed that NCE cannot be achieved in the standard model by non-interactive protocols, i.e., protocols sending only one message per encryption. Canetti, Halevi, and Katz [CHK05a] circumvent Nielsen's impossibility result by periodically changing decryption keys and requiring a small amount of interaction between senders, namely, the current time period that has to be communicated.

The main concern is that such interactive, i.e., non-local, constructions are not suitable for all use cases. Examples include scenarios where the sender and receiver are not on-line simultaneously, as is for instance the case for encrypted email which are typically stored on an IMAP server and are retrieved by the receiver at any time, possibly when the sender is no longer available. Indeed, many real-world communications are "fire-and-forget," where additional communication is impossible or too expensive, e.g., for UDP, sensor networks, etc.

Security in the random-oracle model [BR93] can be a reasonable price to pay for adaptive non-interactive encryption, especially when other building blocks in a composed protocol rely on random oracles already. Nielsen [Nie02] provided a non-interactive construction in the random-oracle model, but presented it as a secure message transmission (SMT) protocol, which lets a sender send a secure message to a receiver but does not provide the environment any actual ciphertexts, which is the same as in the work done by Choi et al. [CDMW09]. This is problematic when a higher-level protocol needs to perform further operations on ciphertexts, such as signing, hashing, or re-encrypting ciphertexts. Another problem is that Nielsen's construction assumes

authenticated channels between the sender and receiver. When the underlying encryption scheme is used without authenticated channels, the scheme becomes malleable and thus does not satisfy the security properties that one expects.

5.1.1 Contribution

This chapter presents a new non-committing encryption functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ that gives the encrypting party access to actual ciphertexts, much like Canetti's $\mathcal{F}_{\mathsf{PKE}}$ and Canetti-Halevi-Katz' $\mathcal{F}_{\mathsf{AFSE}}$, but unlike Nielsen's $\mathcal{F}_{\mathsf{SMT}}$. It also presents a new game-based notion FULL-SIM that is proven to be equivalent to the ideal functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, and to be strictly stronger than most existing notions, e.g., by Hazay et al. [HPW15], but incomparable to Canetti et al.'s notion [CHK05a]. Apart from acting as a sanity check for the UC functionality, the game-based notion is also easier to prove schemes secure. It is then shown that an existing encryption scheme [CLNS16, Nie02, BR93] (See also Chapter 4) based on trapdoor one-way permutations satisfies FULL-SIM, and hence securely implements $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ against adaptive adversaries, in the random-oracle model. Secure erasures are not required, which is important in practice because realizing secure erasures is virtually impossible on modern hardware and software, as intermediate results may be copied or moved across the RAM, swap partitions, and SSD memory blocks.

The efficient and simple scheme presented in the prior chapter is a reminiscent of Bellare-Rogaway's IND-CCA-secure encryption scheme [BR93] as well as Nielsen's SMT scheme. A public key is a trapdoor permutation f, its corresponding secret key the trapdoor f^{-1}. Slightly simplified, the scheme encrypts a message m by choosing a random element x from the domain of f and produces the ciphertext $(f(x), \mathcal{H}(x) \oplus m, \mathcal{H}(x, m))$ where $\mathcal{H}$ is a random oracle. To decrypt a ciphertext $(c_{(1)}, c_{(2)}, c_{(3)})$, one computes $x' \leftarrow f^{-1}(c_{(1)})$, $m' \leftarrow c_{(2)} \oplus \mathcal{H}(x')$, and checks that $c_{(3)} = \mathcal{H}(x', m')$.

Because an instantiation of the $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ functionality immediately gives rise to a non-interactive secure message transmission protocol, Nielsen's impossibility result extends to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, meaning that $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ cannot be securely instantiated in the standard model. Moreover, Canetti et al. [CJS14] pointed out that a globally accessible and programmable random oracle (or common reference string, for that matter) cannot be instantiated with a single hash function.

Recommending a provably uninstantiable scheme for use in cryptographic protocols is, of course, controversial. Most other examples of uninstantiable random-oracle schemes [CGH04, GK03] are contrived constructions that would never be considered for real-world use, if only because there actually exist efficient alternatives that still have a chance of being instantiated securely. The quite natural construction clearly doesn't fall in this category. Rather, one can compare it to highly practical random-oracle schemes, such as RSA-OAEP encryption [BR94] or Schnorr signatures [Sch91], that are widely used in spite of strong indications that they may not be instantiable [BF05, PV05]. Moreover, these schemes are used in spite of the fact that reasonably efficient standard-model alternatives do exist for encryption and signatures. As Nielsen's result showed, the same is not true for non-interactive encryption, even if one were willing to make sacrifices on efficiency.

Security in the random-oracle model, even if uninstantiable, is still meaningful, in the sense that it does protect against "generic" adversaries that treat the hash function as a black box. Moreover, a proof in the random-oracle model is still a good "sanity check" to exclude other design flaws in the protocol and is obviously highly preferable to not having a security proof at

all.

Therefore, in scenarios where interaction is simply not an option, and where security in the local random-oracle model is acceptable, the functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ can very conveniently be used as a building block in higher-level protocols. This is demonstrated with the example of signcryption, showing that the generic encrypt-then-sign construction from the game-based world also works in UC, yielding the first adaptively UC-secure signcryption scheme without erasures. In other words, the formalization presented allows, for the first time, to analyze protocols relying on such a building for the first time.

5.1.2 Related Work

The first non-committing encryption schemes [BH92, CFGN96] were only able to encrypt single bits and require new public keys to be distributed for each encrypted bit. Unsurprisingly, in the light of Nielsen's impossibility result [Nie02], most "two-sided" NCE schemes, i.e., where both sender and receiver can be corrupted at any time, are interactive [Bea97, BH92, CEN15, CFGN96, CHK05a, CDMW09, DN00, GWZ09, HLP15, HP17, HORR16, HOR15, LCC06, Nie02, ZB10], meaning that either encryption and decryption cannot be done locally without additional communication, there is an a-priori upper bound on the number of ciphertexts created before a new key pair has to be generated, or there is no ciphertext at all.

Nielsen [Nie02] and the prior chapter present non-interactive schemes in the random-oracle model without this a priori bound. However, as discussed earlier, the former is presented as an SMT protocol and therefore not easily reusable as a UC building block; also, it implicitly assumes authenticated channels between parties. The latter scheme is the same as the construction as in the prior chapter, but is proven secure under a game-based (i.e., non-UC) definition and assumes secure erasures in the analysis. This chapter provides a UC analysis of the scheme without assuming secure erasures, and tighter security.

Single-sided definitions of NCE have also been proposed, where only the sender *or* the receiver can be corrupted [BDWY12, BHK12, FHKW10, FHKP16, HPW15, HJR16, HRW16, HR14, JL00], but not both at the same time. On the upside of these definitions is that there are realizations in the standard, i.e., non-random-oracle model, while some still resort to idealized models [HP16]. More importantly, these constructions are not proven secure if both sides can be corrupted.

5.1.3 Additional Preliminaries

In this section, some additional definitions are presented. For the following definitions, the following additional conventions are used. A variable $\mathbf{v}$ in bold face is a vector. $\mathcal{I}$ is an index set. The notation $\mathbf{v}_{\mathcal{I}}$ means that the elements in $\mathbf{v}$ are indexed by the index set $\mathcal{I}$. Also, the random oracle is not made explicit in these definitions, as the definitions are geared to be secure in the standard model.

5.1.3.1 SSIM-SO-Security

The following definition is plainly taken from [HPW15].

Definition 5.1 (SSIM-SO-Security)**.** *An encryption scheme* ENC *is SSIM-SO-secure if for all PPT adversaries $\mathcal{A}$ there exists a stateful PPT simulator S such that for every binary-output*

Experiment $\mathsf{SSIM\text{-}SO\text{-}ideal}^{\mathsf{ENC}}_{S,n}(\lambda)$:
$\mathsf{Dist} \xleftarrow{\$} S(1^\lambda)$
$\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{\$} \mathsf{Dist}_{1,2,\ldots,n}$
$\mathcal{I} \xleftarrow{\$} S()$
$\mathsf{output} \xleftarrow{\$} S(\mathbf{m}_\mathcal{I})$
return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Experiment $\mathsf{SSIM\text{-}SO\text{-}real}^{\mathsf{ENC}}_{S,n}(\lambda)$:
$(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$
$(\mathsf{Dist}, \mathsf{state}_1) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\mathsf{ENC}})$
$\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{\$} \mathsf{Dist}_{1,2,\ldots,n}$
$(\mathbf{c}; \mathbf{r}) = ((c_1; r_1), (c_2; r_2), \ldots$
$\quad (c_n; r_n)) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m_i)_{1,2,\ldots,n}$
$(\mathcal{I}, \mathsf{state}_2) \xleftarrow{\$} \mathcal{A}(\mathbf{c}, \mathsf{state}_1)$
$\mathsf{output} \xleftarrow{\$} \mathcal{A}(\mathbf{r}_\mathcal{I}, \mathbf{m}_\mathcal{I}, \mathsf{state}_2)$
return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Figure 5.1: Experiments SSIM-SO-ideal and SSIM-SO-real for the SSIM-SO definition

distinguisher D and any n polynomial in λ it holds that:

$$\left| \Pr[D(\mathsf{SSIM\text{-}SO\text{-}ideal}^{\mathsf{ENC}}_{S,n}(\lambda)) = 1] - \Pr[D(\mathsf{SSIM\text{-}SO\text{-}real}^{\mathsf{ENC}}_{S,n}(\lambda)) = 1] \right| \leq \nu(\lambda)$$

for some negligible function ϵ and the experiments of Figure 5.1.

In a nutshell, this definition requires that an adversary cannot decide whether it sees simulated ciphertexts or real ones, even it sees the randomness used to generate the ciphertexts at some point.

5.1.3.2 RSIM-SO-Security

The following definition is taken from [HPW15], but adjusted for the used notation. Note, here the secret key also contains the randomness used to create it.

Definition 5.2 (RSIM-SO-Security)**.** *An encryption scheme* ENC *is said to be RSIM-SO-secure if for all PPT adversaries $\mathcal{A}$ there exists a stateful PPT simulator S such that for every binary-output distinguisher D and any n polynomial in λ it holds that:*

$$\left| \Pr[D(\mathsf{RSIM\text{-}SO\text{-}ideal}^{\mathsf{ENC}}_{S,n}(\lambda)) = 1] - \Pr[D(\mathsf{RSIM\text{-}SO\text{-}real}^{\mathsf{ENC}}_{S,n}(\lambda)) = 1] \right| \leq \nu(\lambda)$$

for some negligible function ν and the experiments of Figure 5.2.

In a nutshell, this definition requires that an adversary cannot decide whether it sees simulated ciphertexts or real ones, even it sees the secret decryption key (with randomness) at some point. The adversary does never receive any randomness used for encryptions.

5.1.3.3 IND-NCER-Security

The following definition is taken from [CHK05a, HPW15], but explicit state was added to the adversary.

Next, let $\mathsf{ENC}^* = (\mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Enc}^*_{\mathsf{ENC}}, \mathcal{A}\mathsf{Dec}_{\mathsf{ENC}}, \mathsf{Open}^*_{\mathsf{ENC}})$. Algorithm $\mathsf{KeyGen}_{\mathsf{ENC}}$, $\mathsf{Enc}_{\mathsf{ENC}}$, and $\mathsf{Dec}_{\mathsf{ENC}}$ are a standard encryption scheme (without labels). The fake encryption scheme $\mathsf{Enc}^*_{\mathsf{ENC}}$ outputs a ciphertext c^* and a trapdoor t. Given the secret key $\mathsf{sk}_{\mathsf{ENC}}$, the public key $\mathsf{pk}_{\mathsf{ENC}}$, the fake-ciphertext c^*, the trapdoor t, and a plaintext m, algorithm $\mathsf{Open}^*_{\mathsf{ENC}}$ outputs $\mathsf{sk}^*_{\mathsf{ENC}}$. Refer to [HPW15] for a formal definition, including correctness.

Experiment $\mathsf{RSIM\text{-}SO\text{-}ideal}^{\mathsf{ENC}}_{S,n}(\lambda)$:
$\mathsf{Dist} \xleftarrow{\$} S(1^\lambda)$
$\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{\$} \mathsf{Dist}_{1,2,\ldots,n}$
$\mathcal{I} \xleftarrow{\$} S()$
$\mathsf{output} \xleftarrow{\$} S(\mathbf{m}_\mathcal{I})$
return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Experiment $\mathsf{RSIM\text{-}SO\text{-}real}^{\mathsf{ENC}}_{S,n}(\lambda)$:
$(\mathbf{pk}, \mathbf{sk}; \mathbf{r}) = ((\mathsf{pk}_{\mathsf{ENC},1}, \mathsf{sk}_{\mathsf{ENC},1}; r_1) \ldots,$
$(\mathsf{pk}_{\mathsf{ENC},n}, \mathsf{sk}_{\mathsf{ENC},n}; r_n)) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)_{1,2,\ldots,n}$
$(\mathsf{Dist}, \mathsf{state}_1) \xleftarrow{\$} \mathcal{A}(\mathbf{pk})$
$\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{\$} \mathsf{Dist}_{1,2,\ldots,n}$
$\mathbf{c} = (c_1, c_2, \ldots, c_n) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC},i}, m_i)_{1,2,\ldots,n}$
$(\mathcal{I}, \mathsf{state}_2) \xleftarrow{\$} \mathcal{A}(\mathbf{c}, \mathsf{state}_1)$
$\mathsf{output} \xleftarrow{\$} \mathcal{A}(\mathbf{r}_\mathcal{I}, \mathbf{m}_\mathcal{I}, \mathsf{state}_2)$
return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Figure 5.2: Experiments RSIM-SO-ideal and RSIM-SO-real for the RSIM-SO definition

Experiment $\mathsf{IND\text{-}NCER}^{\mathsf{ENC}^*}_{\mathcal{A}}(\lambda)$:
$(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC},0}; r_{\mathsf{key},0}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$(m, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\mathsf{ENC}})$
$c_0 \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m)$
$(c_1, t) \xleftarrow{\$} \mathsf{Enc}^*_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, 1^{|m|})$
$(\mathsf{sk}_{\mathsf{ENC},1}, r_{\mathsf{key}_1}) \xleftarrow{\$} \mathsf{Open}^*_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC},0}, \mathsf{pk}_{\mathsf{ENC}}, c_1, t, m)$
$b^* \xleftarrow{\$} \mathcal{A}(\mathsf{state}, (\mathsf{sk}_{\mathsf{ENC},b}, r_{\mathsf{key}_b}), c_b)$
return 1, if $b^* = b$
return 0

Figure 5.3: ENC^* IND-NCER-Security

Definition 5.3 (IND-NCER-Security)**.** *An encryption scheme* ENC^* *is IND-NCER-secure, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{IND\text{-}NCER}^{\mathsf{ENC}^*}_{\mathcal{A}}(\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

for the experiment given in Figure 5.3.

This definition requires that an adversary $\mathcal{A}$ cannot decide whether it sees simulated secret key randomness (and ciphertext), or the real one, even it receives an encryption of a message of its own choice.

5.2 Game-Based Non-Committing Encryption

Most existing game-based security notions aim at standard-model instantiations and therefore have to circumvent Nielsen's impossibility result [Nie02]. They either do so by encrypting only a single message under each public key (e.g., Canetti, Halevi, and Katz' IND-NCER, i.e.,

non- committing encryption for the receiver, notion [CHK05a]) or by considering only sender corruptions but no receiver corruptions (e.g., the SSIM-SO, i.e., sender-simulatable selective-opening, notion [HPW15]), or only receiver corruptions but no sender corruptions (e.g., the RSIM-SO, i.e., receiver-simulatable selective-opening, notion [HPW15], and the RECV-SIM notion introduced in Chapter 4) allows unlimited ciphertexts and receiver corruptions, but, because it focuses on a setting with secure erasures, does not give the adversary access to the randomness used in encryption or key generation.

This section first introduces a new game-based security notion termed FULL-SIM for non-committing encryption. Unlike existing game-based notions [HPW15], the FULL-SIM adversary simultaneously has access to the randomness used in previous ciphertexts as well as to the randomness used to generate the key pair (and hence, the secret decryption key $\mathsf{sk}_{\mathsf{ENC}}$). Then, a FULL-SIM-secure NCE scheme based on trapdoor one-way permutations in the random-oracle model is given. Finally, the relationship of FULL-SIM to existing notions is proven, finding FULL-SIM to be either strictly stronger or incomparable.

5.2.1 Game-Based Definition of Non-Committing Encryption

A labeled non-committing encryption scheme $\mathsf{ENC} = \{\mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Dec}_{\mathsf{ENC}}\}$ consists of three algorithms. The first algorithm, as for standard encryption schemes, is the key generation algorithm, i.e., $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$, which outputs a public and the corresponding secret key. The public key implicitly specifies a message space $\mathcal{MS}$. The encryption algorithm $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell)$ computes a ciphertext c on input of a public key $\mathsf{pk}_{\mathsf{ENC}}$, a message $m \in \mathcal{MS}$ and a label $\ell \in \{0,1\}^*$. The deterministic decryption algorithm $m' \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, \ell)$ takes as input a secret key $\mathsf{sk}_{\mathsf{ENC}}$, a ciphertext c and a label ℓ and outputs either a message m', or $\perp$ if decryption failed. Clearly, for definitions and schemes without labels, one can simply fix all labels to the empty string below.

The scheme must be correct, meaning that for all $\lambda \in \mathbb{N}$, all $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$, all messages $m \in \mathcal{MS}$, and all labels $\ell \in \{0,1\}^*$, it holds that $m = \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell), \ell)$ with probability one.

Clearly, this is still the standard definition of an encryption scheme, as defined in Chapter 2.

Now the notion of FULL-SIM-security for labeled non-committing encryption schemes ENC is defined. While not made explicit in the notation, $\mathsf{SIM}_{\mathsf{NCE}}$ is allowed to keep state between invocations. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle, i.e., a truly random function chosen fresh at the beginning of the experiment [BR93].

Definition 5.4 (FULL-SIM-Security)**.** *An encryption scheme* ENC *is FULL-SIM-secure if for all PPT adversaries* $\mathcal{A}$ *with binary output there exists a stateful PPT simulator* $\mathsf{SIM}_{\mathsf{NCE}}$ *such that:*

$$\Big|\Pr[\textit{FULL-SIM}\text{-}\mathsf{real}(\lambda) = 1] - \Pr[\textit{FULL-SIM}\text{-}\mathsf{ideal}(\lambda) = 1]\Big| \leq \nu(\lambda)$$

for some negligible function ν*, and the experiments depicted in Figure 5.4. The input* ENCRYPTL *means that the simulator only receives the length of the message in question, while in the case* ENCRYPTM *the simulator receives the message itself, i.e., when one of the participants is already corrupted.*

The definition says that an encryption scheme ENC is FULL-SIM-secure, if no PPT adversary $\mathcal{A}$ can distinguish between simulated ciphertexts and real ones. The adversary $\mathcal{A}$ receives full

Experiment $\mathsf{FULL\text{-}SIM\text{-}ideal}^{\mathsf{ENC}}_{\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}},\mathcal{L}}(\lambda)$:
$\mathsf{pk}_{\mathsf{ENC}} \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{PUBLICKEY}, 1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
return $\mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot),\mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}(\cdot,\cdot)},\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}(\cdot,\cdot)}}_{\mathcal{O}^{\mathsf{GetKey}(\cdot,\cdot)},\mathcal{O}^{\mathsf{GetRand}(\cdot,\cdot)}}(\mathsf{pk}_{\mathsf{ENC}})$
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input m and ℓ:
if oracle $\mathsf{GetKey}()$ has been called:
let $c \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPTM}, m, \ell)$
else:
let $c \xleftarrow{\$} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ENCRYPTL}, \mathcal{L}(m), \ell)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c, m, \ell)\}$
return c
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input c and ℓ:
if $(c, m, \ell) \in \mathcal{Q}$, return m
else, return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{DECRYPT}, c, \ell)$
where oracle $\mathcal{H}$ on input s:
return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, s)$
where oracle $\mathsf{GetKey}()$:
return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$
where oracle $\mathsf{GetRand}(\cdot,\cdot)$ on input (c, ℓ):
if $(c, m, \ell) \notin \mathcal{Q}$ for some m, return $\perp$
else, return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{RAND}, (c, m, \ell))$

Experiment $\mathsf{FULL\text{-}SIM\text{-}real}^{\mathsf{ENC}}_{\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}}}(\lambda)$:
$(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}; r_{\mathsf{key}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
return $\mathcal{A}^{\mathcal{O}^{\mathcal{H}}(\cdot),\mathcal{O}^{\mathsf{Dec}_{\mathsf{ENC}}(\cdot,\cdot)},\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}(\cdot,\cdot)}}_{\mathcal{O}^{\mathsf{GetKey}(\cdot,\cdot)},\mathcal{O}^{\mathsf{GetRand}(\cdot,\cdot)}}(\mathsf{pk}_{\mathsf{ENC}})$
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input m and ℓ:
let $(c; r) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c, m, \ell, r)\}$
return c
where oracle $\mathsf{Dec}_{\mathsf{ENC}}$ on input c and ℓ:
return $\mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, \ell)$
where oracle $\mathcal{H}$ on input s:
return $\mathcal{H}(s)$
where $\mathsf{GetKey}()$:
return r_{key}
where $\mathsf{GetRand}(\cdot,\cdot)$ on input (c, ℓ):
if $(c, m, \ell, r) \notin \mathcal{Q}$, for some r and m,
return $\perp$
else, return r

Figure 5.4: Experiments FULL-SIM-ideal and FULL-SIM-real for the FULL-SIM definition

adaptive access to oracles for new encryptions, decryptions, the randomness used for encryptions, as well as the randomness used for generating the secret key. The simulated ciphertexts do not contain any information about the plaintext other than what is explicitly given to the simulator by the leakage function $\mathcal{L}$. It is defined that the leakage function $\mathcal{L} : \{0,1\}^* \to \mathbb{N}_0$ returns the (bit-)length of the message m in question, which is a reasonable leakage definition for the given use case. Only when the adversary asks for the randomness used to generate a ciphertext or the secret key, does the simulator obtain the corresponding messages, upon which it must provide a consistent view to $\mathcal{A}$.

5.2.2 Instantiation

Now a concrete instantiation for an encryption scheme ENC that is FULL-SIM-secure is given. The construction is identical to the encryption scheme presented in the prior chapter, which, in turn, borrows ideas from Bellare and Rogaway [BR93] and Nielsen [Nie02]. It is recalled here and proven secure under the stronger FULL-SIM notion.

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function, modeled as a random oracle. Further, it requires an encoding scheme $\mathsf{EC} = (\mathsf{ec}, \mathsf{dc})$ which allows to map arbitrary length messages to a list of blocks with fixed length. More precisely, let $\mathsf{ec} : \{0,1\}^* \to (\{0,1\}^\lambda)^+$ be a *deterministic* injective encoding function and $\mathsf{dc} : (\{0,1\}^\lambda)^+ \to \{0,1\}^*$ be the corresponding deterministic decoding function that returns $\perp$ if no valid pre-image exists. It is required that both functions

are computable in polynomial time and that the output length of ec only depends on the length of its input, while dc and ec need to be perfectly correct, i.e., for all $\lambda \in \mathbb{N}$, and all messages $m \in \{0,1\}^*$ it holds that $m = \mathsf{dc}(\mathsf{ec}(m))$ with probability one, just as done in the prior Chapter.

Construction. Here, the construction from Chapter 4 is given again for convenience. It is practical enough for use in real protocols, which avoids the major obstacle for real-life deployment. Hence, the complexity is hidden inside the simulator $\mathsf{SIM}_{\mathsf{NCE}}$, which is given in the proof.

Construction 5.5 (FULL-SIM-secure ENC)**.** *Construct* ENC *in the following way.*

$\mathsf{KeyGen}_{\mathsf{ENC}}$**.** *Generate the key pair in the following way:*

1. *Generate a random* TDP *, i.e.,* $(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$*. The message space* $\mathcal{MS}$ *is* $\{0,1\}^*$*.*
2. *Output the public key* $\mathsf{pk}_{\mathsf{ENC}} = (f, \Sigma)$*, and* $\mathsf{sk}_{\mathsf{ENC}} = f^{-1}$ *as the secret key.*

$\mathsf{Enc}_{\mathsf{ENC}}$**.** *To encrypt a message* m *w.r.t. to* $\mathsf{pk}_{\mathsf{ENC}}$ *and label* ℓ *do:*

1. *Let* $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{ec}(m)$*.*
2. *Draw* $x \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(\Sigma)$*, compute* $c_1 \leftarrow f(x)$*,* $c_2^i \leftarrow \mathcal{G}(i,x) \oplus m_i$ *for* $i = 1, 2, \ldots, k$*, and* $c_3 \leftarrow \mathcal{K}(x, k, m, \ell)$*.*
3. *Output the ciphertext* $c \leftarrow (c_{(1)}, (c_{(2)}^1, c_{(2)}^2 \ldots, c_{(2)}^k), c_{(3)})$*.*

$\mathsf{Dec}_{\mathsf{ENC}}$**.** *To decrypt a ciphertext* c *w.r.t. to* $\mathsf{sk}_{\mathsf{ENC}}$ *and label* ℓ *do:*

1. *Parse* c *as* $(c_{(1)}, (c_{(2)}^1, c_{(2)}^2, \ldots, c_{(2)}^{k'}), c_{(3)})$ *for some* $k' \geq 1$*.*
2. *Compute* $x' \leftarrow f^{-1}(c_{(1)})$ *and* $m_i' \leftarrow \mathcal{G}(i, x') \oplus c_{(2)}^i$ *for* $i = 1, 2, \ldots, k'$*.*
3. *Let* $m' \leftarrow \mathsf{dc}(m_1', \ldots, m'_{k'})$*.*
4. *If* $m' = \perp$ *or* $c_{(3)} \neq \mathcal{K}(x', k', m', \ell)$*, output* $\perp$*. Output* m'*.*

The above construction clearly fulfills perfect correctness, if EC is perfectly correct.

5.2.3 Security of The Construction

Now is proven that the above construction is actually FULL-SIM-secure.

Theorem 5.6. *The construction* $\mathsf{ENC} = \{\mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Dec}_{\mathsf{ENC}}\}$ *above is FULL-SIM-secure, if* $\mathsf{KeyGen}_{\mathsf{TDP}}$ *is a secure trapdoor permutation generator and if* $\mathcal{H}$ *is modeled as a fully programmable, and observable, random oracle.*

Proof. The proof is providing a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ such that the view created by the simulator is indistinguishable from the view provided by the real experiment. The general idea is that the simulator honestly generates the key pair, and honestly samples the first part $c_{(1)}$ of all ciphertexts c, but draws $c_{(2,i)}$ and $c_{(3)}$ randomly. Once the plaintext corresponding to a simulated ciphertext becomes known, i.e., when the adversary makes a GetRand or GetKey query, the simulator programs the random oracle $\mathcal{H}$ such that the ciphertext actually decrypts to the correct message. As all the randomness used by the simulator is drawn honestly, this can

be simulated as well, so secure erasures are not required. The only thing that can make the simulation fail, is that the programming of the random oracle fails because entries have already been assigned. It is shown that such an event gives rise to an algorithm breaking the one-wayness of the TDP.

Description of the Simulator. The simulator $\mathsf{SIM}_{\mathsf{NCE}}$ keeps two initially empty lists $L_{\mathcal{H}}$, and $L_{\mathcal{R}}$. The first list $L_{\mathcal{H}}$ contains pairs (s, h), where h is the value returned by the random oracle on the query $\mathcal{H}(s)$. The second list $L_{\mathcal{R}}$ stores entries (c, m, ℓ, r, x) to keep track of the relationship between (simulated and non-simulated) ciphertexts c, messages m, labels ℓ, sampling randomness r and sampled values x.

Key Generation. On input of $(\mathsf{PUBLICKEY}, 1^\lambda)$, $\mathsf{SIM}_{\mathsf{NCE}}$ honestly generates the key pair, i.e., it generates $((f, f^{-1}, \Sigma); r_{\mathsf{key}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$. It stores $\mathsf{sk}_{\mathsf{ENC}} = f^{-1}$, and r_{key}. $\mathsf{SIM}_{\mathsf{NCE}}$ returns (f, Σ) as $\mathsf{pk}_{\mathsf{ENC}}$.

Random-Oracle Queries. For each query $(\mathsf{ROQUERY}, s)$, $\mathsf{SIM}_{\mathsf{NCE}}$ checks whether there is an entry $(s, h) \in L_{\mathcal{H}}$ for some h. If so, $\mathsf{SIM}_{\mathsf{NCE}}$ returns h. Else, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $h \xleftarrow{\$} \{0,1\}^\lambda$, adds (s, h) to $L_{\mathcal{H}}$, and returns h.

Encryption. Depending on the type of input, the simulator needs to branch:

- On input $(\mathsf{ENCRYPTM}, m, \ell)$, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ encrypts honestly, i.e., it runs $(c; r) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell)$, using $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, s)$ for random-oracle calls $\mathcal{H}(s)$, adds $(c, m, \ell, r, \perp_x)$ to $L_{\mathcal{R}}$, and returns c.
- On input $(\mathsf{ENCRYPTL}, \mathcal{L}(m) = |m|, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $(x; r) \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(1^\lambda)$. Let $k \leftarrow \frac{|\mathsf{ec}(1^{|m|})|}{\lambda}$. It sets $c_{(1)} \leftarrow f(x)$ and chooses $c_{(2,i)} \xleftarrow{\$} \{0,1\}^\lambda$ for $1 \le i \le k$ and $c_{(3)} \xleftarrow{\$} \{0,1\}^\lambda$. It adds $(c, \perp_m, \ell, r, x)$ to $L_{\mathcal{R}}$ and returns c.

Decryption. On input of $(\mathsf{DECRYPT}, c, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ needs to provide a plaintext. It computes $x' \leftarrow f^{-1}(c_{(1)})$, lets $m'_i \leftarrow c_{(2,i)} \oplus \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, (i, x'))$ for $i = 1, 2, \ldots, k$, and computes $m' \leftarrow \mathsf{dc}(m'_1, m'_2, \ldots, m'_k)$. If $m' = \perp$ or $c_{(3)} \neq \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, (x', k, m', \ell))$, it returns $\perp$, otherwise it returns m'. Note that $\mathsf{SIM}_{\mathsf{NCE}}$ never receives a request where it has to decrypt a simulated ciphertext.

Encryption Randomness. On input of $(\mathsf{RAND}, (c, m, \ell))$, look up a tuple $(c, m', \ell, r, x) \in L_{\mathcal{R}}$ where $m' \in \{m, \perp_m\}$. Note that, from the description of experiment $\mathsf{FULL\text{-}SIM\text{-}ideal}$, such a tuple always exists. If $m' = m$, then $\mathsf{SIM}_{\mathsf{NCE}}$ simply returns r. If $m' = \perp_m$, then $\mathsf{SIM}_{\mathsf{NCE}}$ must first program $\mathcal{H}$ to ensure consistent encryption, as with the given randomness r the adversary can re-encrypt m to see whether it obtains c. It does so as follows. Let $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{EC.ec}(m)$. If $((x, k, m, \ell), c'_{(3)}) \in L_{\mathcal{H}}$ for some $c'_{(3)}$ or if $((i, x), h_i) \in L_{\mathcal{H}}$ for some $1 \le i \le k$, then event BAD happened and $\mathsf{SIM}_{\mathsf{NCE}}$ aborts. Otherwise, $\mathsf{SIM}_{\mathsf{NCE}}$ adds $((i, x), m_i \oplus c_{(2,i)})$ for $1 \le i \le k$ as well as $((x, k, m, \ell), c_{(3)})$ to $L_{\mathcal{H}}$. It then updates $(c, \perp_m, \ell, r, x)$ to (c, m, ℓ, r, x) in $L_{\mathcal{R}}$ and returns r.

Key Leakage. On input $(\mathsf{KEYLEAK}, \mathcal{Q})$, $\mathsf{SIM}_{\mathsf{NCE}}$ first programs the random oracle $\mathcal{H}$ to ensure consistent decryption for all $(c, m, \ell) \in Q$ in the same way as for answering $\mathsf{GetRand}$

queries above. $\mathsf{SIM}_{\mathsf{NCE}}$ then returns the random coins r_{key} used to generate the secret key $\mathsf{sk}_{\mathsf{ENC}}$.

From the way random oracles are programmed, it is clear that the simulation is perfect unless the event BAD happens. Using $\Pr[\mathsf{REAL}]$ and $\Pr[\mathsf{IDEAL}]$ as shorthand notations for the probability that the experiment outputs 1 for the real and ideal experiments, respectively, it holds that

$$\begin{aligned}
&\Big|\Pr[\mathsf{REAL}] - \Pr[\mathsf{IDEAL}]\Big| \\
&\quad= \Big|(\Pr[\mathsf{REAL} \mid \mathsf{BAD}] - \Pr[\mathsf{IDEAL} \mid \mathsf{BAD}]) \cdot \Pr[\mathsf{BAD}] \\
&\qquad + (\Pr[\mathsf{REAL} \mid \overline{\mathsf{BAD}}] - \Pr[\mathsf{IDEAL} \mid \overline{\mathsf{BAD}}]) \cdot \Pr[\overline{\mathsf{BAD}}]\Big| \\
&\quad= \Big|(\Pr[\mathsf{REAL} \mid \mathsf{BAD}] - \Pr[\mathsf{IDEAL} \mid \mathsf{BAD}])\Big| \cdot \Pr[\mathsf{BAD}] \qquad (5.1) \\
&\quad\leq \Pr[\mathsf{BAD}]\ , \qquad (5.2)
\end{aligned}$$

where (5.1) is true because $\Pr[\mathsf{REAL} \mid \overline{\mathsf{BAD}}] - \Pr[\mathsf{IDEAL} \mid \overline{\mathsf{BAD}}] = 0$ and (5.2) is true because the first factor of (5.1) is at most one.

Reduction from Trapdoor Permutations. It remains to prove that the event BAD happens with negligible probability. This is done by proving that any adversary that causes BAD to occur gives rise to an algorithm breaking the one-wayness of the trapdoor permutation. The reduction is similar to the analysis of the prior chapter, but is tighter and explicitly gives out the randomness used for encryption to the adversary.

Let q_e be the number of encryption queries, q_d the number of decryption queries, and q_h the number of random-oracle queries to $\mathcal{H}$. Assume towards contradiction that $\Pr[\mathsf{BAD}] > \nu(\lambda)$. One can then construct an algorithm $\mathcal{B}$ which outputs the preimage of a TDP challenge point y with non-negligible probability. Algorithm $\mathcal{B}$ receives (f, y, Σ) as input from the TDP challenger. It then interacts with $\mathcal{A}$ as follows.

Key Generation. On input of $(\mathsf{PUBLICKEY}, 1^\lambda)$, $\mathcal{B}$ draws a random index $j \xleftarrow{\$} [1, q_e]$, and returns (f, Σ) as $\mathsf{pk}_{\mathsf{ENC}}$.

RO Queries. On input $(\mathsf{ROQUERY}, s)$, $\mathcal{B}$ checks if there is an entry $(s, h) \in L_{\mathcal{H}}$ for some h. If so, $\mathcal{B}$ returns h. Else, $\mathcal{B}$ draws $h \xleftarrow{\$} \{0,1\}^\lambda$.

If s is of the form (x, k, m, ℓ) and there exists a previously rejected ciphertext that, by assigning h as the output of $\mathcal{H}(s)$, should have been considered valid, then it is said that event BADH happened and $\mathcal{B}$ aborts. More precisely, if there exists a tuple $(c, \ell) \in \mathcal{L}_c$ with $c = (c_{(1)}, c_{(1,2)}, \ldots, c_{(k,2)}, c_{(3)})$ such that $c_{(1)} = f(x)$, $c_{(3)} = h$, and $m = \mathsf{dc}(c_{(1,2)} \oplus \mathcal{H}(1, x), \ldots, c_{(k,2)} \oplus \mathcal{H}(k, x))$, then BADH happened and $\mathcal{B}$ aborts, whereby random-oracle queries $\mathcal{H}(i, x)$ are simulated as described here.

Otherwise, $\mathcal{B}$ adds (s, h) to $L_{\mathcal{H}}$ and returns h.

Encryption. On input $(\mathsf{ENCRYPTL}, |m|, \ell)$, it proceeds as follows. If this is the jth encryption query, then $\mathcal{B}$ sets $c_{(1)} \leftarrow y$ and $x, r \leftarrow \perp$. Otherwise, it draws $(x; r) \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(1^\lambda)$,

sets $c_{(1)} \leftarrow f(x)$, and tests whether $x \in \mathcal{L}_x$. If so, then it is said that event BADX happened, and $\mathcal{B}$ aborts, otherwise $\mathcal{B}$ adds x to $\mathcal{L}_x$. Let $k \leftarrow \frac{|\mathsf{EC.ec}(1^{|m|})}{\lambda}$. Algorithm $\mathcal{B}$ chooses $c_{(2,i)} \xleftarrow{\$} \{0,1\}^\lambda$ for all $1 \le i \le k$ and $c_{(3)} \xleftarrow{\$} \{0,1\}^\lambda$, adds $(c, \perp_m, \ell, r, x)$ to $L_{\mathcal{R}}$, and returns c. Note, $(\mathsf{ENCRYPTM}, m, \ell)$ is never received.

Decryption. On input $(\mathsf{DECRYPT}, c, \ell)$ from $\mathcal{A}$, $\mathcal{B}$ proceeds as follows. Look for an entry $((x, k, m, \ell), c_{(3)}) \in \mathcal{L}_{\mathcal{H}}$ such that $f(x) = c_{(1)}$ and $m = \mathsf{EC.dc}(m_1, m_2, \ldots, m_k)$ for $m_i \leftarrow c_{(2,i)} \oplus \mathcal{H}(i, x)$, where calls to $\mathcal{H}$ are simulated as above. Note that at most one such entry can exist because, TDP being a permutation, there exists only one $x \in \Sigma$ such that $f(x) = c_{(1)}$, which then unique defines $m_1, \ldots, m_k$ and thereby m. If no such entry exists, then $\mathcal{B}$ adds (c, ℓ) to $\mathcal{L}_c$ and returns $\perp$, otherwise it returns m.

Encryption Randomness. On input of $(\mathsf{RAND}, (c, m, \ell))$, look up a tuple $(c, m', \ell, r, x) \in L_{\mathcal{R}}$ for $m' \in \{m, \perp_m\}$. If $m' = m$ then $\mathcal{B}$ returns r. If $m' = \perp_m$ then $\mathcal{B}$ must program $\mathcal{H}$ to ensure consistent encryption. If now the conditions of the BAD event in $\mathsf{SIM_{NCE}}$ are satisfied, i.e., if there exists a tuple $((x', k, m, \ell), \cdot) \in L_{\mathcal{H}}$ or a tuple $((i, x'), \cdot) \in L_{\mathcal{H}}$ such that $f(x') = c_{(1)}$, and if additionally c is the j-th simulated ciphertext, i.e., if $x = r = \perp$, then $\mathcal{B}$ outputs x' as its preimage for $y = c_{(1)}$. If the conditions for BAD are satisfied but c is not the j-th ciphertext, then $\mathcal{B}$ aborts. If the conditions for BAD are not satisfied, then $\mathcal{B}$ programs the random oracle in the same way as $\mathsf{SIM_{NCE}}$, i.e., by adding $((i, x), m_i \oplus c_{(2,i)})$ for $1 \le i \le k$ as well as $((x, k, m, \ell), c_{(3)})$ to $L_{\mathcal{H}}$. Algorithm $\mathcal{B}$ updates $(c, \perp_m, \ell, r, x)$ to (c, m, ℓ, r, x) in $L_{\mathcal{R}}$ and returns r.

Key Leakage. On input of $(\mathsf{KEYLEAK}, \mathcal{Q})$, $\mathcal{B}$ checks each entry (i, x) and (x, k, m, ℓ) in $\mathcal{L}_{\mathcal{H}}$ whether $f(x) = y$. If so, then $\mathcal{B}$ returns x as the preimage of y to the challenger. Otherwise, it aborts.

Algorithm $\mathcal{B}$ succeeds in inverting y with probability $\frac{1}{q_e}$ whenever event BAD happens and neither BADH nor BADX happen, i.e.

$$\begin{aligned} \Pr[\mathcal{B} \text{ succeeds}] &= \frac{1}{q_e} \Pr[\mathsf{BAD} \wedge \overline{\mathsf{BADH}} \wedge \overline{\mathsf{BADX}}] \\ &\ge \frac{1}{q_e}\Big(\Pr[\mathsf{BAD}] - \Pr[\mathsf{BADH}] - \Pr[\mathsf{BADX}]\Big) \end{aligned}$$

Note that not being able to respond to a $\mathsf{KEYLEAK}$ input does not influence the success probability of $\mathcal{B}$, because after a $\mathsf{KEYLEAK}$ input the BAD event can no longer happen anyway.

The event BADH happens when the response to a new random-oracle query $\mathcal{H}(x, k, m, \ell)$ turns a ciphertext that was previously rejected during a decryption query into a valid ciphertext. For each decryption query $\mathsf{Dec_{ENC}}(c, \ell)$, there is only a single random-oracle query $\mathcal{H}(x, k, m, \ell')$ that could cause BADH to happen though, namely the query where $f(x) = c_{(1)}$, k is the number of blocks in $c_{(2)} = (c_{(1,2)}, \ldots, c_{(k,2)})$, $m = \mathsf{dc}(c_{(1,2)} \oplus \mathcal{H}(1, x), \ldots, c_{(k,2)} \oplus \mathcal{H}(k, x))$, and $\ell' = \ell$. Each of those random-oracle queries $\mathcal{H}(x, k, m, \ell')$ has probability $\frac{1}{2^\lambda}$ to hit $c_{(3)}$, so the overall probability of BADH is at most:

$$\Pr[\mathsf{BADH}] \le \frac{q_d}{2^\lambda}$$

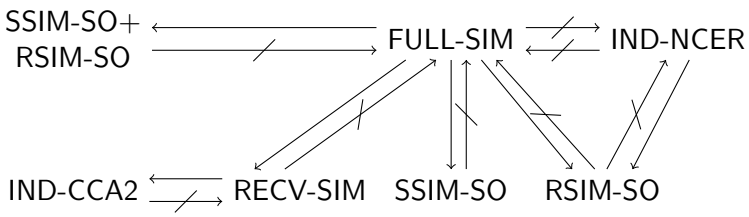

Figure 5.5: Implications of security definitions. Solid arrows denote strict implications, while striked out arrows denote separations.

The event BADX happens when during encryption, a randomly chosen value $x \xleftarrow{\$} \Sigma$ hits one of the at most q_e elements of $\mathcal{L}_{\mathcal{R}}$. Since $|\Sigma| \geq 2^{2\lambda}$, the probability of this happening is

$$\Pr[\mathsf{BADX}] \leq \tfrac{q_e^2}{2^{2\lambda}}$$

Putting everything together, if $\nu'(t)$ is the maximum advantage of a PPT algorithm to break the one-wayness of TDP, then one has that

$$|\Pr[\mathsf{REAL}] - \Pr[\mathsf{IDEAL}]| \;\leq\; q_e\nu'(t) + \frac{q_d}{2^\lambda} + \frac{q_e^2}{2^{2\lambda}}$$

which proves the theorem. □

It is stressed that Nielsen's underlying construction [Nie02] is not a FULL-SIM-secure encryption scheme, as it is trivially malleable. Refer to Appendix H for his construction.

The following corollary immediately follows from the construction.

Corollary 5.7. *If trapdoor one-way permutations exist, then there also exist FULL-SIM-secure encryption schemes in the fully programmable, and observable, random-oracle model without secure erasures.*

5.3 Relationships Between Security Notions

As depicted in Figure 5.5, it is now shown that the definition is strictly stronger than the existing definitions RECV-SIM given in Chapter 4, SSIM-SO [HPW15], and RSIM-SO [HPW15], while being incomparable to IND-NCER [CHK05a, HPW15]. Now, these statements are proven.

Clearly, the RECV-SIM given in the prior chapter is the basis of the FULL-SIM notion, but does not give the adversary access to encryption or key generation randomness. As one would expect, the new notion implies RECV-SIM, and also IND-CCA2-security, it automatically follows that FULL-SIM security also implies IND-CCA2-security.

Theorem 5.8 (FULL-SIM $\implies$ RECV-SIM)**.** *Any encryption scheme that is FULL-SIM secure is also RECV-SIM secure.*

Proof. Assume there is an adversary $\mathcal{A}$ which can guess whether it is run in RECV-SIM-real, or RECV-SIM-ideal resp., with a probability non-negligibly better than $\frac{1}{2}$. One can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to distinguish between FULL-SIM-real and FULL-SIM-ideal with the same probability.

$\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{ENC}}$ from its own challenger. It then initializes $\mathcal{A}$ with $\mathsf{pk}_{\mathsf{ENC}}$. Then, for every query s to $\mathcal{H}$ from $\mathcal{A}$, $\mathcal{B}$ forwards the query to its own oracle $\mathcal{H}$, and returns the result unmodified to $\mathcal{A}$. Likewise, for every query (m, ℓ) to $\mathsf{Enc}_{\mathsf{ENC}}$, $\mathcal{B}$ asks the encryption oracle provided by its own challenger. Again, the answer is passed to $\mathcal{A}$ unmodified. Finally, every query (c, ℓ) to the decryption is also honestly answered using the decryption oracle provided to $\mathcal{B}$. Eventually, $\mathcal{A}$ returns $\mathsf{state}_{\mathcal{A}}$. $\mathcal{B}$ saves $\mathsf{state}_{\mathcal{A}}$. $\mathcal{B}$ then asks its own oracle GetKey to receive r_{key}. $\mathcal{B}$ then generates the corresponding secret key of $\mathsf{pk}_{\mathsf{ENC}}$ by letting $(\mathsf{pk}'_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda; r_{\mathsf{key}})$. $\mathcal{B}$ then continues simulating $\mathcal{A}$ with input $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{state}_{\mathcal{A}})$. The random oracle queries made by $\mathcal{A}$ are answered as before. Finally, $\mathcal{A}$ will return its guess b^*. $\mathcal{B}$ uses b^* as its own guess. Clearly, $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, as $\mathcal{B}$ can perfectly simulate $\mathcal{A}$'s environment. □

Next, the chosen-plaintext definitions given by Hazay, Patra, and Warinschi [HPW15] are considered. As RSIM-SO and SSIM-SO do not incorporate labels, it is defined that the encryption and decryption oracles only accept empty labels, as the notions are essentially equivalent [SG02].

First, RSIM-SO security is addressed.

Theorem 5.9 (FULL-SIM $\implies$ RSIM-SO)**.** *Any public-key encryption scheme that is FULL-SIM secure, is also RSIM-SO secure.*

Proof. Let $\mathcal{A}$ be an adversary, together with a distinguisher D, which together guess in which RSIM-SO-experiment it is run in with a probability non-negligibly better than $\frac{1}{2}$. One can then construct an adversary $\mathcal{B}$ which can distinguish between FULL-SIM-real and FULL-SIM-ideal with a non-negligible probability.

This statement is proven by a series of hybrids. Let Exp_0 be RSIM-SO-ideal, while Exp_n is the experiment RSIM-SO-real. Also, define Exp_i such that the first i public keys are simulated, while the remaining $n - i$ are honestly generated. Further assume towards contradiction that there is an adversary $\mathcal{A}$ that can distinguish between Exp_i, and Exp_{i+1} for some index i. One can then construct an adversary $\mathcal{B}$ which can break the FULL-SIM-security definition. In particular, $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{ENC}}$ from its own challenger. It then embeds $\mathsf{pk}_{\mathsf{ENC}}$ as $\mathsf{pk}_{\mathsf{ENC}i+1}$, and leaves the other $\mathsf{pk}_{\mathsf{ENC}}$s untouched, and gives the complete public key vector to $\mathcal{A}$. Algorithm $\mathcal{B}$ then samples the message vector $\mathbf{m}$ according to the received distribution Dist. It generates each c_j, $j \neq i$ according to the current experiment, but asks its own challenger to receive the ciphertext c_i. The ciphertext vector $\mathbf{c}$ is then given to $\mathcal{A}$. Eventually, $\mathcal{A}$ returns $\mathcal{I}$, and $\mathcal{B}$ then receives all the randomness used to create the corresponding secret keys, and the messages $\mathbf{m}_{\mathcal{I}}$, which it provides to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs output, which $\mathcal{B}$, together with $\mathbf{m}$, and $\mathcal{I}$ to D. Whatever D then outputs, is then also output by $\mathcal{B}$. Clearly, the probability that $\mathcal{B}$ can successfully distinguish between FULL-SIM-real, and FULL-SIM-ideal is thus non-negligible. □

Theorem 5.10 (FULL-SIM $\implies$ SSIM-SO)**.** *A public-key encryption scheme that is FULL-SIM secure, is also SSIM-SO secure.*

Proof. Assume there is a distinguisher D with some arbitrary, but fixed adversary $\mathcal{A}$ which together can decide whether they run in SSIM-SO-real or SSIM-SO-ideal with a probability non-negligibly better than $\frac{1}{2}$. Then an adversary $\mathcal{B}$ can be constructed which distinguishes between FULL-SIM-real and FULL-SIM-ideal with the same probability.

In the first step, the random oracle is rewired to $\mathcal{B}$'s own random oracle. $\mathcal{B}$ then receives the challenge public key $\mathsf{pk}_{\mathsf{ENC}}$. $\mathsf{pk}_{\mathsf{ENC}}$ is simply passed to $\mathcal{A}$ to initialize the adversary. $\mathcal{B}$ receives Dist and state_1 from $\mathcal{A}$. $\mathcal{B}$ then samples the message vector $\mathbf{m}$ according to the received Dist. Each m_i is then sent to $\mathcal{B}$'s own encryption oracle to receive each ciphertext c_i. Let $\mathbf{c}$ denote the complete vector of the ciphertexts. $\mathcal{A}$ is then given state_1 and $\mathbf{c}$. Eventually, $\mathcal{A}$ outputs $(\mathcal{I}, \mathsf{state}_2)$. For each $i \in \mathcal{I}$, $\mathcal{B}$ then queries its own oracle $\mathsf{GetRand}(\cdot)$ with c_i to receive each r_i. Let the vector of all received r_i be $\mathbf{r}$. $\mathcal{A}$ is then given $(\mathbf{r}, \mathbf{m}_{\mathcal{I}}, \mathsf{state}_2)$. Finally, $\mathcal{A}$ returns output. D is given $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$. Whatever D outputs, is also output by $\mathcal{B}$. Clearly, as one can perfectly simulate the environment of $\mathcal{A}$ and D, $\mathcal{B}$ success probability equals the one of $\mathcal{A}$ with D. □

Next, some additional results to show the separations in the other direction, i.e., to show that the FULL-SIM definition is strictly stronger than RECV-SIM, RSIM-SO, and SSIM-SO are needed.

Theorem 5.11. *There is no FULL-SIM-secure NCE in the standard model.*

Proof. This follows by construction. In Section 5.5 is shown how to realize the round-optimal secure message transfer functionality $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ using the functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, which is black-box realized by any FULL-SIM-secure encryption scheme. The theorem follows by plugging in the impossibility result by Nielsen [Nie02]. □

The implications above are strict. This follows from the fact that there are standard model instantiations of RSIM-SO-secure, and SSIM-SO-secure, schemes.

This also means that one cannot avoid the programmability of the random oracle in the construction, which also follows from the results given by Nielsen [Nie02].

Theorem 5.12. *If one-way trapdoor permutations exist, and the Decisional composite residuosity [Pai99] (DCR) assumption holds, then IND-NCER security is incomparable to FULL-SIM security in the random oracle model. Meaning, there exists a scheme that is IND-NCER secure but not FULL-SIM secure and vice versa.*

Proof. For the first direction, note that there are constructions in the standard model, i.e., under the DCR-Assumption, for IND-NCER security [CHK05a], but none for FULL-SIM security, as proven before. Thus, IND-NCER security does not imply FULL-SIM security.

For the other direction, it is shown that there is a FULL-SIM-secure construction, which is not IND-NCER-secure. Namely, one already knows that the presented construction is FULL-SIM-secure, if TDPs exist in the random oracle model. Now is shown that the construction is not IND-NCER-secure.

Let $\mathsf{Open}_{\mathsf{ENC}}^{*}$, and $\mathsf{Enc}_{\mathsf{ENC}}^{*}$ be arbitrarily defined, while the remaining algorithms are defined as in the construction. In particular, $\mathcal{A}$ draws a random message $m \xleftarrow{\$} \{0,1\}^{\lambda}$. Then, the challenger (Note, $\mathsf{Open}_{\mathsf{ENC}}^{*}$ and $\mathsf{Enc}_{\mathsf{ENC}}^{*}$ are public and thus no random oracle programming is possible!) needs to return a ciphertext c, which correctly decrypts to m, i.e., at least $c_3 = \mathcal{H}(x, k, m, \ell)$ must hold. The probability that $\mathsf{Enc}_{\mathsf{ENC}}^{*}$ guesses the messages correctly upfront (and therefore the correct output) is negligible, i.e., at most $\frac{q_h}{2^{\lambda}}$, where q_h is the number of random oracle queries. The other case is similar, i.e., that c_3 was drawn randomly, and one hopes that the unique random oracle query (x, k, m, ℓ) (Note, k, ℓ, and x are fixed), makes the ciphertext valid.

Clearly, this is negligible as well. Thus, the probability that $\mathsf{Open}^*_{\mathsf{ENC}}$ returns randomness for the secret key such that the ciphertext returned decrypts correctly is negligible ($\frac{1}{2^\lambda}$), regardless of the choice of $\mathsf{Open}^*_{\mathsf{ENC}}$, and $\mathsf{Enc}^*_{\mathsf{ENC}}$. It thus follows that the probability that this happens is equal/less than $\frac{q_h+1}{2^\lambda}$, which is negligible. □

Moreover, even SSIM-SO-Security and RSIM-SO-Security together do not imply FULL-SIM-Security, as the adversary cannot proceed adaptively in the RSIM-SO game, i.e., the distribution is fixed, which is not the case in the FULL-SIM-Security game.

Theorem 5.13 (RECV-SIM $\not\Longrightarrow$ FULL-SIM)**.** *If perfectly binding commitments exist, then there exists a scheme that is RECV-SIM secure but not FULL-SIM secure.*

Proof. The idea of the proof is as follows. If there exists is a perfectly-binding (bit) commitment-scheme, then RECV-SIM-Security does not imply FULL-SIM-Security. With perfectly-binding is meant that even a computationally unbounded adversary can find only one way to open a given commitment c w.r.t. to the (even adversarially chosen) public parameters $\mathsf{pp}_{\mathsf{CC}}$.

Let ENC be any FULL-SIM-secure encryption, and $\mathsf{Commit}_{\mathsf{CC}}$ be a perfectly-binding commitment-scheme as defined.

Now construct the counterexample $\mathsf{ENC}' = \{\mathsf{KeyGen}'_{\mathsf{ENC}}, \mathsf{Enc}'_{\mathsf{ENC}}, \mathsf{Dec}'_{\mathsf{ENC}}\}$ as follows, such that is only RECV-SIM-secure, but not FULL-SIM-secure.

Construction 5.14 (Counterexample ENC′)**.** *A counterexample* ENC′ *can be constructed as follows.*

$\mathsf{KeyGen}'_{\mathsf{ENC}}$**.** *To generate a key pair do:*

1. *Generate* $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{ENC.KeyGen}_{\mathsf{ENC}}(1^\lambda)$.
2. *Return* $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}})$.

$\mathsf{Enc}'_{\mathsf{ENC}}$**.** *To encrypt a message m w.r.t. to* $\mathsf{pk}_{\mathsf{ENC}}$ *and label ℓ do:*

1. *Generate* $\mathsf{pp}_{\mathsf{CC}} \xleftarrow{\$} \mathsf{ParGen}_{\mathsf{CC}}(1^\lambda)$.
2. *For each bit* $b_i \in m$*, let* $(c_i, r_i) \xleftarrow{\$} \mathsf{Commit}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, b_i)$.
3. *Set* $\ell' \leftarrow (\ell, \mathsf{pp}_{\mathsf{CC}}, (c_1, c_2, \ldots, c_{|m|}))$.
4. *Let* $c \xleftarrow{\$} \mathsf{ENC.Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m, \ell')$.
5. *Return* $(c, \mathsf{pp}_{\mathsf{CC}}, (c_1, c_2, \ldots, c_{|m|}))$.

$\mathsf{Dec}'_{\mathsf{ENC}}$**.** *To decrypt a ciphertext c w.r.t. to* $\mathsf{sk}_{\mathsf{ENC}}$ *and label ℓ do:*

1. *Parse c as* $(c, \mathsf{pp}_{\mathsf{CC}}, (c_1, c_2, \ldots, c_{|m|}))$.
2. *Let* $\ell' \leftarrow (\ell, \mathsf{pp}_{\mathsf{CC}}, (c_1, c_2, \ldots, c_{|m|}))$.
3. *Return* $\mathsf{ENC.Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, \ell')$.

It is obvious that the construction is still RECV-SIM-secure by the following argument. If the commitment scheme used is computationally hiding, the ciphertext c is indistinguishable from an encryption using any other message m' of the same length, as the randomnesses used to generate the commitments are never given to the adversary attacking the scheme. Thus, the simulator can choose a random message of the same length.

However, the scheme ENC' cannot be FULL-SIM-secure, as the probability that any simulator $\mathsf{SIM}_{\mathsf{NCE}}$ guesses the correct message is negligible, as the commitment-scheme is perfectly binding, i.e., no simulator can equivocate the commitments. For any other (meaningful) definition of the leakage oracle $\mathcal{L}$, similar arguments exist. □

5.4 Universally Composable Non-Committing Encryption

In this section, an ideal functionality in the UC framework for non-committing encryption $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is introduced. It is shown that $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and FULL-SIM-security are essentially equivalent, in the sense that any FULL-SIM-secure scheme immediately gives rise to a secure instantiation of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$
transmission functionality $\mathcal{F}_{\mathsf{SMT}}$, so that, as an immediate consequence, Nielsen's impossibility result [Nie02] excludes any secure instantiations of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ (and therefore, of FULL-SIM-secure encryption schemes) in the standard model.

5.4.1 Ideal Functionality for Non-Committing Encryption

The ideal functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is depicted in Figure 5.6. In a nutshell, $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ encapsulates *local* public-key encryption, which is one of the most basic operations in modern cryptography. The functionality is based on the $\mathcal{F}_{\mathsf{AFSE}}$ functionality of Canetti et al. [CHK05a], but without the a-priori bound on the ciphertexts to be generated, while it also supports labels. It therefore neither needs any update interfaces, nor any special corruption interfaces. Compared to the definition by Canetti et al. [CKN03], it is also enforced that once a ciphertext is decrypted, the decryption remains fixed.

Encryption and decryption in the functionality are local operations that generate and decrypt actual ciphertexts. The ciphertexts are provided by the adversary $\mathcal{A}$ which, as long as the owner of the key pair is honest, only receives the information explicitly provided by the leakage function $\mathcal{L}$. The notation "send x to $\mathcal{A}$ and wait for y from $\mathcal{A}$" is used as a shorthand notation for requests to responsive environments as defined by Camenisch et al. [CEK+16], so that the functionality "stalls" until $\mathcal{A}$ provides a response y through a dedicated interface, i.e., the adversary is forced to provide an answer right away. While the functionality waits for a message from $\mathcal{A}$, the adversary cannot invoke any other interfaces of the ideal functionality, generate any network traffic, or activate or corrupt parties.

Previous functionalities in the literature require the adversary to provide an encryption algorithm that the functionality runs to generate ciphertexts for the encrypted messages [Can01, CH06, KT08, KT09, KT11]. All realizations using this paradigm suffer from the selective decommitment problem [Can01, KT09] or inherently require static adversaries. This problem is avoided by querying the adversary for each ciphertext that needs to be generated *and* decrypted.

The public key for an instance of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is also provided by the adversary. To avoid implying certified keys, the encryption interface also accepts queries for different public keys than the registered key of this instance. Decryption for simulated ciphertexts works as expected, in the sense that for every ciphertext generated by the encryption interface, the functionality returns

1. `Key Generation.` On input (KEYGEN, sid) from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$ or a record $(\mathsf{key\text{-}rec}, \cdot)$ exists, ignore.
   - Send (KEYGEN, sid) to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, \mathsf{pk})$ from $\mathcal{A}$.
   - If $\mathsf{pk} = \bot$, ignore.
   - Create record $(\mathsf{key\text{-}rec}, \mathsf{pk})$ and output $(\mathsf{KEYCONF}, sid, \mathsf{pk})$ to $\mathcal{P}$.
2. `Encryption.` On input $(\mathsf{ENCRYPT}, sid, \mathsf{pk}, m, \ell)$ from party $\mathcal{Q}$:
   - If $\mathsf{pk} = \bot$, ignore.
   - If no record $(\mathsf{key\text{-}rec}, \mathsf{pk})$ exists, send $(\mathsf{ENCRYPTM}, sid, \mathsf{pk}, m, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - Else, if $sid = (\mathcal{P}, sid')$ and $\mathcal{P}$ is corrupt, send $(\mathsf{ENCRYPTM}, sid, \mathsf{pk}, m, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - Else, send $(\mathsf{ENCRYPTL}, sid, \mathsf{pk}, \mathcal{L}(m), \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - If there is a record $(\mathsf{key\text{-}rec}, v)$, let $\mathsf{pk}' \leftarrow v$, else let $\mathsf{pk}' \leftarrow \bot$.
   - If there is a record $(\mathsf{enc\text{-}rec}, sid, \mathsf{pk}, \cdot, \ell, c)$ and $\mathsf{pk} = \mathsf{pk}'$, ignore.
   - If there is a record $(\mathsf{dec\text{-}rec}, sid, \cdot, \ell, c)$ and $\mathsf{pk} = \mathsf{pk}'$, ignore.
   - Create record $(\mathsf{enc\text{-}rec}, sid, \mathsf{pk}, m, \ell, c)$.
   - Output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk})$ to $\mathcal{Q}$.
3. `Decryption.` On input $(\mathsf{DECRYPT}, sid, c, \ell)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$, ignore.
   - If no record $(\mathsf{key\text{-}rec}, \mathsf{pk})$ exists, ignore.
   - If a record $(\mathsf{enc\text{-}rec}, sid, \mathsf{pk}, m, \ell, c)$ exists, output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.
   - If a record $(\mathsf{dec\text{-}rec}, sid, m, \ell, c)$ exists, output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.
   - Send $(\mathsf{DECRYPT}, sid, c, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{PLAINTEXT}, sid, m)$ from $\mathcal{A}$.
   - Create record $(\mathsf{dec\text{-}rec}, sid, m, \ell, c)$.
   - Output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.

Figure 5.6: Ideal encryption functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. $\mathcal{L}$ is a leakage function.

the original message without any involvement of the adversary. If, however, a ciphertext was not honestly generated, the functionality asks the adversary to provide the decryption.

The Interfaces. Next, briefly explained, the interfaces of the $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ functionality.

- The KEYGEN interface can only be called once and allows the key-pair owner $\mathcal{P}$ to generate a key pair. The public key $\mathsf{pk}_{\mathsf{ENC}}$ is provided by the adversary.
- The ENCRYPT interface allows any party, including the key-pair owner itself, to encrypt a message of arbitrary length. If the public key that is given as part of the input equals the public key stored for this instance, then the adversary only receives the information explicitly given by the leakage function $\mathcal{L}$, as long as $\mathcal{P}$ is honest. Otherwise, the functionality does not give any security guarantees, and gives the adversary the plaintext. Note, the adversary does not learn which party wants to encrypt, while the provided pk may also be adversarially chosen. The adversary must provide fresh ciphertexts for the stored public key and cannot overwrite ciphertexts.
- The DECRYPT interface allows the key-pair owner to decrypt a given ciphertext. For simulated ciphertexts and ciphertexts that were decrypted before, the corresponding plaintexts are directly returned by the functionality. All other ciphertexts are sent to the adversary which needs to provide a decryption. Decryption is consistent, in the sense that once a ciphertext is mapped to a given plaintext, the result of decrypting that ciphertext will always be the same.

5.4.2 Instantiation of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$

The construction is a wrapper around any FULL-SIM-secure labeled non-committing encryption scheme $\mathsf{ENC} = (\mathsf{KeyGen}_{\mathsf{ENC}}, \mathsf{Enc}_{\mathsf{ENC}}, \mathsf{Dec}_{\mathsf{ENC}})$ to match the input and output behavior of the ideal functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, using the label to bind a ciphertext to a session *sid*. Any calls that ENC makes to a random oracle are relayed to an instance of the random-oracle functionality $\mathcal{F}_{\mathsf{RO}}$ with session identifier $(sid, \mathcal{F}_{\mathsf{RO}})$. Note, one cannot rely on the more realistic global random oracles [CJS14], because the instantiation cannot avoid programmability, as proven in Section 5.3.

Key Generation. On input of (KEYGEN, sid), check that $sid = (\mathcal{P}, sid')$, and $\mathcal{P}$ the identity. If this is not the case, ignore. If a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}})$ exists, ignore. Generate $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$, store $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}})$, and output $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{ENC}})$.

Encryption. On input of $(\mathsf{ENCRYPT}, sid, \mathsf{pk}, m, \ell)$, ignore if $\mathsf{pk} = \perp$, or $sid \neq (\mathcal{P}, sid')$. Generate $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}, m, (\ell, sid))$. Output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk})$.

Decryption. On input of $(\mathsf{DECRYPT}, sid, c, \ell)$, check that $sid = (\mathcal{P}, sid')$. If this is not the case, or no record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}})$ exists, ignore. Otherwise, let $m \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c, (\ell, sid))$. Output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

Theorem 5.15. *If* ENC *is FULL-SIM secure, then the above protocol securely realizes* $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ *in the* $\mathcal{F}_{\mathsf{RO}}$*-hybrid model without secure erasures and with adaptive corruptions.*

Proof. By the FULL-SIM security of ENC, there must exist a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ so that no PPT adversary exists that can distinguish between the real FULL-SIM game and the ideal FULL-SIM game with $\mathsf{SIM}_{\mathsf{NCE}}$ with non-negligible probability. Given such a simulator $\mathsf{SIM}_{\mathsf{NCE}}$, a UC simulator SIM for the above instantiation of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is now described. Subsequently it is shown that any environment that can distinguish whether it is interacting with the real protocol and a real-world adversary $\mathcal{A}$ or with $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and SIM can be used to build a FULL-SIM distinguisher for $\mathsf{SIM}_{\mathsf{NCE}}$, contradicting the FULL-SIM security of ENC.

Simulator. Given a FULL-SIM simulator $\mathsf{SIM}_{\mathsf{NCE}}$ and a real-world UC adversary $\mathcal{A}$, consider the following UC simulator SIM for $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.

Key Generation. On input (KEYGEN, sid), SIM calls the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ with the input $(\mathsf{PUBLICKEY}, 1^\lambda)$, which returns $\mathsf{pk}_{\mathsf{ENC}}$. SIM then records $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}})$ and sends $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{ENC}})$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.

Encryption. On input $(\mathsf{ENCRYPTM}, sid, \mathsf{pk}, m, \ell)$, the simulator SIM computes $(c; r) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}, m, (\ell, sid))$, creates record $(\mathsf{enc\text{-}rec}, sid, c, m, \ell, r, \mathsf{pk}, \mathsf{false})$, and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. On input $(\mathsf{ENCRYPTL}, sid, \mathsf{pk}, \mathcal{L}(m), \ell)$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{ENCRYPTL}, \mathcal{L}(m), (\ell, sid))$ to $\mathsf{SIM}_{\mathsf{NCE}}$ to receive c. It then creates a record $(\mathsf{enc\text{-}rec}, sid, c, \perp_m, \ell, \perp_r, \mathsf{pk}, \mathsf{false})$ and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. Note that this case implies $\mathsf{pk} = \mathsf{pk}_{\mathsf{ENC}}$.

Decryption. On input $(\mathsf{DECRYPT}, sid, c, \ell)$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{DECRYPT}, c, (\ell, sid))$ to obtain m and sends $(\mathsf{DECRYPT}, sid, m)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.

Random oracle. In the $\mathcal{F}_{\mathsf{RO}}$-hybrid model, SIM must also provide responses to all parties' inputs to the $\mathcal{F}_{\mathsf{RO}}$ functionality. On input $(\mathsf{ROQUERY}, sid', s)$ from a party $\mathcal{P}$, SIM outputs $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{ROQUERY}, s)$ if $sid' = (sid, \mathcal{F}_{\mathsf{RO}})$, or runs the actual code of $\mathcal{F}_{\mathsf{RO}}$ otherwise.

Corruptions. The scheme is proven within standard corruption definition [Can01] where, upon corruption of a party, the simulator receives all previous inputs and outputs of that party and needs to provide a consistent view of the real-world state for that party to the real-world adversary $\mathcal{A}$. The simulator needs to consider different cases depending on which party gets corrupted.

If the holder of the secret key $\mathcal{P}$ becomes corrupted, SIM receives six lists corresponding to the inputs and outputs of the three interfaces of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ to and from $\mathcal{P}$.

The simulator SIM has to provide a realistic snapshot of $\mathcal{P}$'s real-world state to the adversary $\mathcal{A}$ that must contain, apart from the list of previous inputs and outputs that SIM just obtained from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, the randomness used in key generation and encryption. To obtain the key generation randomness from $\mathsf{SIM}_{\mathsf{NCE}}$, SIM must compile the set $\mathcal{Q}$ of all honestly generated ciphertexts and corresponding plaintexts (c, m, ℓ) that were encrypted under $\mathsf{pk}_{\mathsf{ENC}}$ by any party (not just $\mathcal{P}$). The list of ciphertexts c and labels ℓ can be looked up in its own records $(\mathsf{enc\text{-}rec}, sid, c, \perp_m, \ell, \perp_r, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{false})$. Since $\mathcal{P}$ is now corrupt, SIM can obtain the corresponding plaintexts m by querying the decryption interface on $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ with $(\mathsf{DECRYPT}, sid, c, \ell)$. After thus composing the list $\mathcal{Q}$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{KEYLEAK}, \mathcal{Q})$ to obtain the key generation randomness r_{key}.

To obtain the randomness used for every encryption under $\mathsf{pk}_{\mathsf{ENC}}$ performed by $\mathcal{P}$, SIM considers $\mathcal{P}$'s previous outputs from the encryption interface $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk}_{\mathsf{ENC}})$ that it received from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. For each of these outputs, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{RAND}, (c, m, (\ell, sid)))$ to obtain the encryption randomness r. For ciphertexts generated by $\mathcal{P}$ under other public keys $\mathsf{pk} \neq \mathsf{pk}_{\mathsf{ENC}}$, SIM also has records of the form $(\mathsf{enc\text{-}rec}, sid, c, m, \ell, r, \mathsf{pk}, \mathsf{false})$, so that SIM can simply give the stored r and update the record to $(\mathsf{enc\text{-}rec}, sid, c, m, \ell, r, \mathsf{pk}, \mathsf{true})$.

The latter is important to ensure that randomness will not be given out for another party later. Namely, when encrypting under an adversarial public key pk, one cannot exclude that different randomness r, r' yield the same ciphertext c. If this happens for two encryption calls made by two different parties, and these parties later get corrupted, then the simulator must ensure that different randomness r and r' is given out to each of these parties. (Recall that the simulator doesn't learn which party initiated an encryption, so it doesn't know which randomness it used for which party.) By flagging a record with true when the randomness was given out, SIM ensures that the same randomness will not be given out again to a different party.

If any other party $\mathcal{Q}$ becomes corrupted, SIM needs to provide to $\mathcal{A}$ all $\mathcal{Q}$'s previous inputs and outputs, as well as the randomness used to create the ciphertexts that $\mathcal{Q}$ generated. For all previous encryption outputs $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk})$ to $\mathcal{Q}$, which SIM now obtains from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, $\mathsf{SIM}_{\mathsf{NCE}}$ is called with $(\mathsf{RAND}, (c, m, (\ell, sid)))$ to return the randomness which SIM simply passes on to $\mathcal{A}$. It also includes the randomnesses used for other public keys $\mathsf{pk} \neq \mathsf{pk}_{\mathsf{ENC}}$, which are all stored in the records $(\mathsf{enc\text{-}rec}, sid, c, m, \ell, r, \mathsf{pk}, \mathsf{false})$.

Reduction From FULL-SIM. Fix an environment and a real-world adversary $\mathcal{A}$ that can distinguish the real protocol with $\mathcal{A}$ from the ideal functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ with simulator SIM. One can then construct an adversary $\mathcal{B}$ that breaks the FULL-SIM-security of the underlying encryption scheme with essentially the same probability (ignoring negligible parts).

Essentially, let $\mathcal{B}$'s inputs and oracles handle the key generation, encryption, decryption, random-oracle, and corruption queries, much like SIM lets these be handled by $\mathsf{SIM}_{\mathsf{NCE}}$. First, $\mathcal{B}$ receives the public key $\mathsf{pk}_{\mathsf{ENC}}$ from the challenger in the FULL-SIM experiment.

If the environment, before the KEYGEN interface is called, instructs an honest party to encrypt a message m by providing it with input $(\mathsf{ENCRYPT}, sid, \mathsf{pk}, m, \ell)$, $\mathcal{B}$ simply calculates $(c; r) \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}, m, (\ell, sid))$, and outputs $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk})$.

It also saves $(c, m, \ell, r, \mathsf{pk})$. If at any point this party becomes corrupted, the reduction simply hands over the corresponding rs stored in $(c, m, \ell, r, \mathsf{pk})$. Decryption queries are simply ignored before the KEYGEN interface is called. Queries to the random-oracle functionality for $sid' = (sid, \mathcal{F}_{\mathsf{RO}})$ are rewired to $\mathcal{B}$'s own random oracle; for other session identifiers, it executes the real code of $\mathcal{F}_{\mathsf{RO}}$.

When at some point the environment instructs the key-pair owner $\mathcal{P}$ to generate its keys by providing input (KEYGEN, sid), then $\mathcal{B}$ embeds the key $\mathsf{pk}_{\mathsf{ENC}}$ in the $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{ENC}})$ output. The encryption queries for a "incorrect" public keys $\mathsf{pk} \neq \mathsf{pk}_{\mathsf{ENC}}$ are answered as before.

However, for an encryption query with $\mathsf{pk} = \mathsf{pk}_{\mathsf{ENC}}$, $\mathcal{B}$ refers to its own encryption oracle, by calling $\mathsf{Enc}_{\mathsf{ENC}}(m, (\ell, sid))$ which returns a ciphertext c.

Store $(c, m, \ell, \perp_r, \mathsf{pk}_{\mathsf{ENC}})$. Output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk}_{\mathsf{ENC}})$. Decryption queries are handled similarly, i.e., on input $(\mathsf{DECRYPT}, sid, c, \ell)$, $\mathcal{B}$ calls $\mathsf{Dec}_{\mathsf{ENC}}(c, (\ell, sid))$ to obtain m and returns $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

If at some point a party becomes corrupted, $\mathcal{B}$ needs to provide a consistent view of its state to $\mathcal{A}$. It uses the records $(c, m, \ell, \perp_r, \mathsf{pk}_{\mathsf{ENC}})$ belonging to the corrupted party to call $\mathsf{GetRand}(c, (\ell, sid))$ to obtain the randomness r, which is passed to $\mathcal{A}$. The randomness used for "incorrect" public keys $\mathsf{pk} \neq \mathsf{pk}_{\mathsf{ENC}}$ is obtained from its records of the form $(c, m, \ell, r, \mathsf{pk})$. If the key-pair holder becomes corrupted, $\mathcal{B}$

additionally calls $\mathsf{GetKey}()$ to receive r_{key} and passes it to $\mathcal{A}$. At some point, the environment then outputs its guess b^* indicating whether it is running in the real world with $\mathcal{A}$ or in the ideal world with SIM. $\mathcal{B}$ returns b^* as its own guess, indicating that it is running in the FULL-SIM-real experiment with the real encryption scheme, or in FULL-SIM-ideal with $\mathsf{SIM}_{\mathsf{NCE}}$.

It is clear that if $\mathcal{B}$ is running in FULL-SIM-real, then the environment's view is exactly as when interacting with the real protocol. Likewise, if $\mathcal{B}$ is running in FULL-SIM-ideal, then all its oracle queries are handled by $\mathsf{SIM}_{\mathsf{NCE}}$, as is also done by the UC simulator SIM. It follows that $\mathcal{B}$'s advantage is negligibly close to that of $\mathcal{A}$. □

Clearly, any FULL-SIM-secure encryption scheme securely realizes $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. To prove the equivalence, one also need to prove the other direction. In a nutshell, now is shown how one can build a FULL-SIM-secure scheme from $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$, while any adversary against the resulting scheme can be used to construct an environment which can be used to distinguish between the real world and the ideal one.

Theorem 5.16 ($\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}} \implies$ FULL-SIM)**.** *Any protocol that securely realizes $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ gives rise to a FULL-SIM-secure non-committing encryption scheme.*

Proof. The construction of the FULL-SIM-secure encryption scheme ENC from an instantiation of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is quite straightforward. First, a choose random, yet correctly structured, *sid*. The key generation algorithm invokes the KEYGEN interface, and returns the resulting public key $\mathsf{pk}_{\mathsf{ENC}}$ as the public key. Encryption, and decryption resp., invoke the ENCRYPT, and DECRYPT resp., interfaces.

For the proof, first construct a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ for FULL-SIM-ideal given a UC-simulator SIM. Essentially, one has to implement the five oracles given in the FULL-SIM-experiment, by translating the answer given by SIM. Namely, for oracle $\mathcal{H}$, the queries received are simply re-routed to the SIM's random-oracle interface. $\mathsf{pk}_{\mathsf{ENC}}$ is obtained by calling SIM with (KEYGEN, sid). SIM answers with $(\mathsf{KEYCONF}, sid, \mathsf{pk})$. This pk is the public key $\mathsf{pk}_{\mathsf{ENC}}$ which is given to the adversary. For encryption (oracle $\mathsf{Enc}_{\mathsf{ENC}}$), SIM is called with $(\mathsf{ENCRYPTL}, sid, \mathsf{pk}_{\mathsf{ENC}}, \mathcal{L}(m), \ell)$ if oracle GetKey has not been queried, or $(\mathsf{ENCRYPTM}, sid, \mathsf{pk}_{\mathsf{ENC}}, m, \ell)$ in all other cases. Here, (m, ℓ) is the corresponding input. In both cases, SIM responds with $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk}_{\mathsf{ENC}})$. This c is given to the adversary. Decryption (oracle $\mathsf{Dec}_{\mathsf{ENC}}$) is similar, i.e., for each query (c, ℓ) to decrypt, and not seen so far, $(\mathsf{DECRYPT}, sid, c, \ell)$ is sent to SIM, which answers with $(\mathsf{PLAINTEXT}, sid, m)$. If (c, ℓ) has been seen before, $\mathsf{SIM}_{\mathsf{NCE}}$ directly returns m as given in the prior corresponding query to SIM. This m is simply given to the adversary. If the oracle GetRand on input (c, ℓ) is called, and (c, ℓ) has never been queried before, $\mathsf{SIM}_{\mathsf{NCE}}$ hands $(\mathsf{ENCRYPT}, sid, \mathsf{pk}_{\mathsf{ENC}}, m, \ell)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk})$ to SIM. SIM then responds with a complete execution history, where the used randomness r is contained. If (c, ℓ) has been seen before, $\mathsf{SIM}_{\mathsf{NCE}}$ looks up the prior corresponding answer from SIM, which contains r. In any case, the randomness r is then given to the adversary. Finally, the oracle GetKey is simulated by handing over all lists of the form (KEYGEN, sid), $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{ENC}})$, $(\mathsf{ENCRYPTL}, sid, \mathsf{pk}_{\mathsf{ENC}}, \mathcal{L}(m), \ell)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk}_{\mathsf{ENC}})$ to SIM, if this oracle has not been called before. In the execution history given to $\mathsf{SIM}_{\mathsf{NCE}}$, there is the randomness r_{key}, which can be returned to the adversary by $\mathsf{SIM}_{\mathsf{NCE}}$. If oracle GetKey has been called before, $\mathsf{SIM}_{\mathsf{NCE}}$ directly hands over r_{key} to the adversary. Clearly, $\mathsf{SIM}_{\mathsf{NCE}}$ perfectly translates all requests.

In the second step, assume that there is an adversary $\mathcal{A}$ which wins against the constructed simulator $\mathsf{SIM}_{\mathsf{NCE}}$. One can then construct an environment $\mathcal{B}$, and an adversary $\mathcal{A}_d$ which together can distinguish between the ideal world with SIM, and the real world. Namely, one proceeds as follows. $\mathcal{B}$ requests a public key $\mathsf{pk}_{\mathsf{ENC}}$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ by sending (KEYGEN, sid) to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ (through the dummy party defined by *sid*). The received $\mathsf{pk}_{\mathsf{ENC}}$ is embedded to initialize $\mathcal{A}$, which is run inside the adversary $\mathcal{A}_d$. Every query to the random oracle by $\mathcal{A}$ is also rewired. However, as there are local random oracles, one needs a workaround. Namely, $\mathcal{B}$ directs $\mathcal{A}_d$ to query the random oracle for each query. The returned value is simply given to $\mathcal{A}$. On the ith encryption query for message m_i, $\mathcal{B}$ spawns a new party $\mathcal{P}_i$ which it feeds with $(\mathsf{ENCRYPT}, sid, \mathsf{pk}_{\mathsf{ENC}}, m_i, \ell_i)$. Upon receiving $(\mathsf{CIPHERTEXT}, sid, c_i, m_i, \ell_i, \mathsf{pk}_{\mathsf{ENC}})$, $\mathcal{B}$ records (m_i, c_i, ℓ_i, i), and gives c_i to $\mathcal{A}$. Decryption queries are delegated through the functionality as well. The answer is simply given to $\mathcal{A}$. If $\mathcal{A}$ wants to see randomness of given ciphertext c_i with label ℓ_i, $\mathcal{B}$ looks up record (m_i, c_i, ℓ_i, i), and corrupts $\mathcal{P}_i$. The simulator then returns the randomness r_i somewhere in the execution history, which is then given to $\mathcal{A}$. Likewise, if $\mathcal{A}$ wants to see r_{key}, $\mathcal{B}$ $\mathcal{B}$ corrupts the holder of the secret key (defined by *sid*), which presents the execution history, where r_{key} is stored. Eventually, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. □

5.5 Secure Message Transfer from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$

The functionality $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ (secure message transfer) combines confidentiality and authentication between two entities. Now is shown how to use $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ s.t. it realizes $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ with $\mathcal{F}_{\mathsf{CA}}$ and digital signatures. Using the modular approach, i.e., by using $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, the proof becomes much more readable.

It was decided to adjust the functionality given in [Nie02] (See Figure H.1 in Appendix H) to the version depicted in Figure 5.7. Namely, the functionality requires initialization interfaces, allows multi-message transmission, and explicitly allows the adversary to overrule "hold-back" messages, which was implicit in the realization given in [Nie02]. In a nutshell, the simulator needs to know when it has to generate the keys for the honest parties, which also implies that these *fresh* keys have to be registered with $\mathcal{F}_{\mathsf{CA}}$, as $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ is not using JUC [CR03]. Moreover, using the given instantiation, one is able to completely simulate the execution history, not only the random tapes.

Conventions. It is required that $sid = (\mathcal{P}, \mathcal{Q}, sid')$ for some sid'. The instantiation neither uses $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ nor $\mathcal{F}_{\mathsf{Auth}}$ [Can04], as this section focuses on $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. Of course, they can be used instead of the direct signature construction. Refer to the work done by Gjøsteen and Kråkmo [GK07] for an instantiation with $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$. In a nutshell, the public keys generated by the functionality are simply authenticated by $\mathcal{F}_{\mathsf{CA}}$ or the like.

Explanation. Let is give a high-level description what each interface does, i.e., a more informal explanation is given to make the functionality more readable.

- INIT. This interface allows the environment to setup the keys for the participants. Namely, the keys have to be created at some point; in prior definitions this was not made explicit. However, as there are no secure erasures, the simulator needs to know when to generate randomness.
- SEND. This interface allows a party to send a message m to the other party. The functionality enforces uniqueness of the message IDs (mid), while only the sending party needs to be initialized if it honest.
- RETRIEVE. This interface allows the adversary to trigger reception of the messages sent so far. Clearly, as the adversary can schedule delivery. This is done on a "per message" basis using mid. If the receiving party is honest, it must be initialized. Note, the decryption is not required to be valid, i.e., not $\perp$.
- CORRUPT. This interface allows the adversary to change hold back messages upon corruption. This interface is called if a honest party sent messages which have never been received, i.e., hold back by the adversary. Thus, the adversary, after corruption, can change those pending messages to whatever it wants.

5.5.1 Protocol Description

For simplicity, the label input to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ is set to be $\perp$, which can be interpreted as the empty string. For simplicity, it is assumed that $\mathcal{F}_{\mathsf{CA}}$ is never queried again, if a valid public key was returned.

1. `Initialize.` Upon input (INIT, sid) from a party $\mathcal{P}'$:
   - If $sid \neq (\mathcal{P}', \cdot, sid')$, or $sid \neq (\cdot, \mathcal{P}', sid')$ ignore.
   - If $sid = (\mathcal{P}', \mathcal{P}', sid')$, ignore.
   - If there is a record $(\mathsf{INIT}, sid, \mathcal{P}')$, ignore.
   - Send $(\mathsf{INIT}, sid, \mathcal{P}')$ to $\mathcal{A}$.
2. `Send.` Upon input $(\mathsf{SEND}, sid, mid, m)$ from a party $\mathcal{P}'$:
   - If $\mathcal{P}' \neq \mathcal{P}$ or $\mathcal{P}' \neq \mathcal{Q}$, ignore.
   - If $\mathcal{P}' = \mathcal{P}$, set $\mathcal{R} \leftarrow \mathcal{Q}$, and $\mathcal{R} \leftarrow \mathcal{P}$ otherwise.
   - If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot, \mathcal{R}, \cdot)$, ignore.
   - If $\mathcal{P}'$ is honest, and no record $(\mathsf{INIT}, sid, \mathcal{P}')$ exists, ignore.
   - Create record $(\mathsf{msg\text{-}rec}, sid, mid, m, \mathcal{R}, \perp)$.
   - If $\mathcal{P}$ or $\mathcal{Q}$ is corrupt, send $(\mathsf{SENDM}, sid, mid, \mathcal{P}', m)$ to $\mathcal{A}$.
   - Send $(\mathsf{SENDL}, sid, mid, \mathcal{P}', \mathcal{L}(m))$ to $\mathcal{A}$.
3. `Receive.` Upon input $(\mathsf{RETRIEVE}, sid, mid, \mathcal{P}')$ from adversary $\mathcal{A}$:
   - If there is no record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot, \mathcal{P}', \perp)$, ignore.
   - If $\mathcal{P}'$ is honest, and no record $(\mathsf{INIT}, sid, \mathcal{P}')$ exists, ignore.
   - Update record $(\mathsf{msg\text{-}rec}, sid, mid, m, \mathcal{P}', \perp)$ to $(\mathsf{msg\text{-}rec}, sid, mid, m, \mathcal{P}')$.
   - Output $(\mathsf{RETRIEVE}, sid, mid, m)$ to $\mathcal{P}'$.
4. `Corrupt.` Upon input $(\mathsf{CORRUPT}, sid, mid, \mathcal{P}', m')$ from $\mathcal{A}$:
   - If $\mathcal{P}'$ is not corrupt, ignore.
   - If $\mathcal{P}' = \mathcal{P}$, set $\mathcal{R} \leftarrow \mathcal{Q}$, and $\mathcal{R} \leftarrow \mathcal{P}$ otherwise.
   - If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot, \mathcal{R})$, ignore.
   - Update record $(\mathsf{msg\text{-}rec}, sid, mid, m, \mathcal{R}, \perp)$ to $(\mathsf{msg\text{-}rec}, sid, mid, m', \mathcal{R})$.
   - Output $(\mathsf{RETRIEVE}, sid, mid, m')$ to $\mathcal{R}$.

Figure 5.7: Ideal secure message transfer functionality $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$. $\mathcal{L}$ is the leakage function.

Step 1a. Initialize Party ($\mathcal{P}$):

a) Upon input (INIT, sid), ignore, if a record $(\mathsf{key\text{-}rec}, sid, \cdot, \cdot, \cdot)$ exists.
b) If $sid \neq (\mathcal{P}, \cdot, sid')$, or $sid = (\mathcal{P}, \mathcal{P}, sid')$, ignore.
c) Query $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ to generate the public key of the encryption scheme, i.e., send $(\mathsf{KEYGEN}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
d) Upon receiving $(\mathsf{KEYCONF}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}_{\mathsf{ENC},\mathcal{P}})$, generate a key pair of a digital signature scheme DSIG, i.e., $(\mathsf{pk}_{\mathsf{Sig},\mathcal{P}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{P}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.
e) Send $(\mathsf{REGISTER}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{CA}})), (\mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{P}}))$ to $\mathcal{F}_{\mathsf{CA}}$.

Step 1b. Initialize Party ($\mathcal{Q}$):

a) Upon input (INIT, sid), ignore, if a record $(\mathsf{key\text{-}rec}, sid, \cdot, \cdot, \cdot)$ exists.
b) If $sid \neq (\cdot, \mathcal{Q}, sid')$, or $sid = (\mathcal{Q}, \mathcal{Q}, sid')$, ignore.
c) Query $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ to generate the public key of the encryption scheme, i.e., send $(\mathsf{KEYGEN}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
d) Upon receiving $(\mathsf{KEYCONF}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}_{\mathsf{ENC},\mathcal{Q}})$, generate a key pair of a digital signature scheme DSIG, i.e., $(\mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{Q}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.
e) Send $(\mathsf{REGISTER}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{CA}})), (\mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}))$ to $\mathcal{F}_{\mathsf{CA}}$.

Step 2a. Send Message ($\mathcal{P}$):

a) Upon input $(\mathsf{SEND}, sid, mid, m)$, check that a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{P}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{P}})$ exists. If not, ignore.
b) If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot)$, ignore.
c) Record $(\mathsf{msg\text{-}rec}, sid, mid, m)$.
d) Query $\mathcal{F}_{\mathsf{CA}}$ with $(\mathsf{RETRIEVE}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{CA}})))$.
e) Upon receiving $(\mathsf{RETRIEVECOMPL}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{CA}})), v)$, check that $v = (\mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}) \neq \perp$. If not, ignore.
f) Encrypt m, i.e., send $(\mathsf{ENCRYPT}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, m, \perp)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
g) Upon receiving input $(\mathsf{CIPHERTEXT}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, \perp, \mathsf{pk}_{\mathsf{ENC},\mathcal{Q}})$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, compute $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig},\mathcal{P}}, (sid, c, mid))$.
h) Send $m' = (sid, mid, c, \sigma)$ to $\mathcal{Q}$.

Step 2b. Send Message ($\mathcal{Q}$):

a) Upon input $(\mathsf{SEND}, sid, mid, m)$, check that record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{Q}})$ exists. If not, ignore.
b) If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot)$, ignore.
c) Record $(\mathsf{msg\text{-}rec}, sid, mid, m)$.
d) Query $\mathcal{F}_{\mathsf{CA}}$ with $(\mathsf{RETRIEVE}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{CA}})))$.
e) Upon receiving $(\mathsf{RETRIEVECOMPL}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{CA}})), v)$, check that $v = (\mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{P}}) \neq \perp$. If not, ignore.
f) Encrypt m, i.e., send $(\mathsf{ENCRYPT}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, m, \perp)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
g) Upon receiving $(\mathsf{CIPHERTEXT}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, \perp, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}})$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, compute $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig},\mathcal{Q}}, (sid, c, mid))$.
h) Send $m' = (sid, mid, c, \sigma)$ to $\mathcal{P}$.

Step 3a. Receive Message ($\mathcal{P}$):

a) Upon input receiving $m' = (sid, mid, c, \sigma)$, check that a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{P}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{P}})$ exists. If not, ignore.
b) If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot)$, ignore.
c) Query $\mathcal{F}_{\mathsf{CA}}$ with $(\mathsf{RETRIEVE}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{CA}}))$.
d) Upon receiving $(\mathsf{RETRIEVECOMPL}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{CA}})), v)$, check that $v = (\mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}) \neq \perp$. If not, ignore.
e) Check σ, i.e., $d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}, (sid, c, mid), \sigma)$. If $d = \mathsf{false}$, ignore.
f) Decrypt the ciphertext c, i.e., send $(\mathsf{DECRYPT}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, \perp)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
g) Upon receiving $(\mathsf{PLAINTEXT}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, \perp)$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, record $(\mathsf{msg\text{-}rec}, sid, mid, m)$.
h) Output $(\mathsf{RETRIEVE}, sid, mid, m, \mathcal{Q})$.

Step 3b. Receive Message ($\mathcal{Q}$):

a) Upon input receiving $m' = (sid, mid, c, \sigma)$, check that a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{Q}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{Q}}, \mathsf{sk}_{\mathsf{Sig},\mathcal{Q}})$ exists. If not, ignore.
b) If there is a record $(\mathsf{msg\text{-}rec}, sid, mid, \cdot)$, ignore.
c) Query $\mathcal{F}_{\mathsf{CA}}$ with $(\mathsf{RETRIEVE}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{CA}})))$.

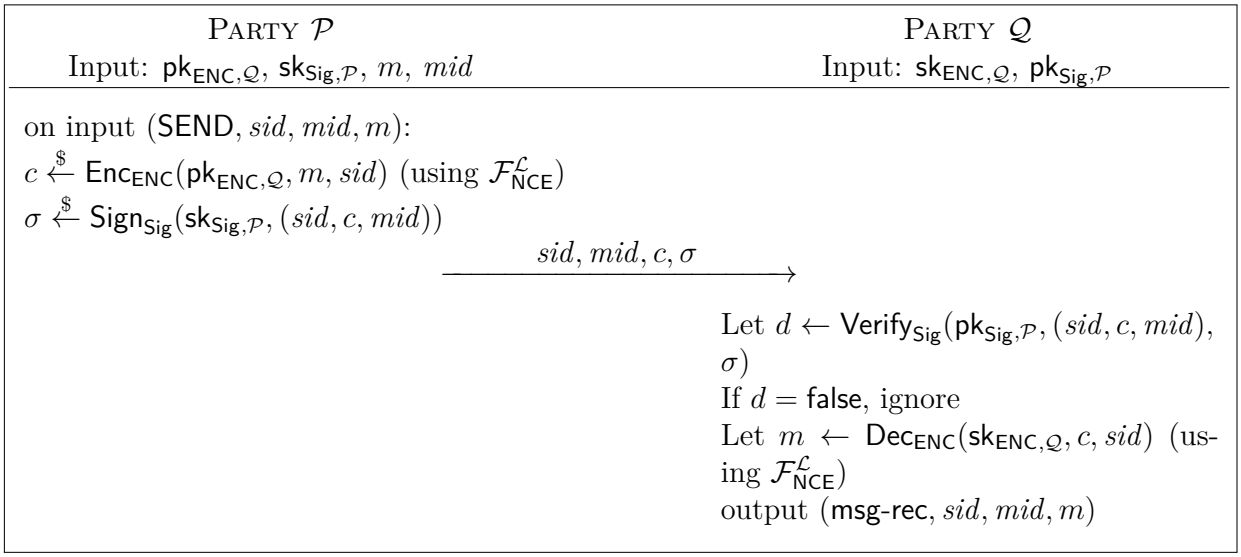

Figure 5.8: Main idea of the secure message transfer from $\mathcal{P}$ to $\mathcal{Q}$

d) Upon receiving $(\mathsf{RETRIEVECOMPL}, (\mathcal{P}, (sid, \mathcal{F}_{\mathsf{CA}})), v)$, check that $v = (\mathsf{pk}_{\mathsf{ENC},\mathcal{P}}, \mathsf{pk}_{\mathsf{Sig},\mathcal{P}}) \neq \perp$. If not, ignore.
e) Check σ, i.e., $d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{Sig},\mathcal{P}}, (sid, c, mid), \sigma)$. If $d = \mathsf{false}$, ignore.
f) Decrypt the ciphertext c, i.e., send $(\mathsf{DECRYPT}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, \perp)$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
g) Upon receiving $(\mathsf{PLAINTEXT}, (\mathcal{Q}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, \perp)$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, record $(\mathsf{msg\text{-}rec}, sid, mid, m)$.
h) Output $(\mathsf{RETRIEVE}, sid, mid, m, \mathcal{P})$.

A simplified overview can be found in Figure 5.8.

5.5.2 Security of the Protocol

Now is proven that the construction actually realizes $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$.

Theorem 5.17. *The above construction securely realizes $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ in the $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})$-Hybrid Model with adaptive corruptions without erasures, if* DSIG *is eUNF-CMA.*

To prove the above theorem, a sequence of games is provided. The corresponding simulator is given afterwards. Namely, a simulator is provided such that the ideal world is indistinguishable from the real world. The main transition is simulating the ciphertexts.

5.5.2.1 Sequence of Games

Game 0: This is the real world, i.e., the simulator runs the protocol for all honest parties.

Game 1: Abort, if a verifying message/signature pair (m^*, σ^*) from the adversary $\mathcal{A}$ is received in the name of a honest party, where the corresponding message m^* has never been signed by a honest party, which visibly changes the behavior. Clearly, a standard reduction yields an adversary which can break the eUNF-CMA security of the used signature scheme DSIG, and thus the difference to the prior game is negligible.

Game 2: Now in the ideal world with simulator SIM is run. This is only an internal change and thus the views do not change.

5.5.2.2 Simulator

Given a $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{CA}})$-hybrid-model adversary $\mathcal{A}$, construct a simulator SIM so that no environment can distinguish running with $\mathcal{A}$ and the real protocol in the $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{CA}})$-hybrid world from running with SIM and $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ in the ideal world in the following way. Note that, apart from interacting correctly with $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$, the simulator must also play the role of the subfunctionalities $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{CA}}$ to $\mathcal{A}$ when they are called by corrupt parties. Essentially, SIM needs to provide network traffic to $\mathcal{A}$ s.t. the simulated messages are indistinguishable from a real protocol. The major obstacle in the simulation is to provide a consistent view upon corruption of a party, which requires heavy book-keeping of the values generated during the simulation. The following simulator is sketched, as the proof should be obvious. Calls to the interfaces of $\mathcal{F}_{\mathsf{CA}}$ are ignored, as they stay untouched, i.e., the code of the functionality is simply run.

High-Level Idea. The simulator proceeds as follows. If both participants are honest, it uses the corresponding "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" (played by $\mathcal{A}$) to generate ciphertexts. As long as both parties are honest, nothing is ever decrypted. The ciphertexts and the corresponding *mid*s are then signed. Upon a corruption, each "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" is provided the corresponding lists compiled by SIM, which then gives out the randomness used for encryptions and secret key generation. The randomness used for the signatures is simulated honestly. If a request is ignored, the execution history is implicit, and given to the adversary upon corruption.

Initialization. Upon receiving $(\mathsf{INIT}, sid, \mathcal{S})$ from $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$, send $(\mathsf{KEYGEN}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{A}$. Upon receiving $(\mathsf{KEYCONF}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}))), \mathsf{pk}_{\mathsf{ENC}})$ from $\mathcal{A}$, create a signature key pair $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}; r_s)$. Create a record $(\mathsf{key\text{-}rec}, \mathsf{pk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}; r_s)$, while "$\mathcal{F}_{\mathsf{CA}}$" is run as is.

Key Generation. Upon receiving $(\mathsf{KEYGEN}, sid', \mathcal{S})$ from $\mathcal{A}$, initialize the party as in the protocol, i.e., ask $\mathcal{A}$ for the public key $\mathsf{pk}_{\mathsf{ENC}}$, returning $(\mathsf{KEYCONF}, sid', \mathsf{pk}_{\mathsf{ENC}})$.

Encryption. Upon receiving $(\mathsf{ENCRYPT}, sid', \mathsf{pk}_{\mathsf{ENC}}{}', m, \ell)$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, asking $\mathcal{A}$ to provide a ciphertext c, creating an encryption record $(\mathsf{enc\text{-}rec}, sid', \mathsf{pk}_{\mathsf{ENC}}{}', m, \ell, c)$, and returning $(\mathsf{CIPHERTEXT}, sid', c, m, \ell, \mathsf{pk}_{\mathsf{ENC}}{}')$.

Decryption. Upon receiving $(\mathsf{DECRYPT}, sid', c, \ell)$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, asking $\mathcal{A}$ for a plaintext m, creating a decryption record $(\mathsf{dec\text{-}rec}, sid', m, \ell, c)$, and returning $(\mathsf{PLAINTEXT}, sid', c, m, \ell)$.

Sending. Upon receiving $(\mathsf{SENDL}, sid, mid, \mathcal{P}', \mathcal{L}(m))$ from $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$, SIM proceeds according to the protocol, but generates the ciphertext according to Game 3 using $\mathcal{A}$. It then creates a record $(\mathsf{SEND}, \mathcal{S}, \perp_m, r_{\mathsf{sign}}, mid, \sigma, c, \perp)$, and sends $m' = (sid, mid, c, \sigma)$ to the party over $\mathcal{A}$.

If $(\mathsf{SENDM}, sid, mid, \mathcal{P}', m)$ is received from $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$, everything is done honestly, i.e., according to the protocol.

Receiving. Now is shown how receiving is handled.

Both Participants Honest. If both participants are honest, the simulator SIM receives control if the recipient "$\mathcal{R}$" receives a message $m = (sid, mid, c, \sigma)$ from $\mathcal{A}$. This message is ignored, if there is no record $(\mathsf{SEND}, \mathcal{S}, \perp_m, r_{\mathsf{sign}}, mid, \cdot, c, \perp)$, is not correctly formatted, or σ is not valid. "$\mathcal{R}$" proceeds honestly, with one notable exception: the ciphertext is never decrypted. If "$\mathcal{R}$" outputs $(\mathsf{RETRIEVE}, sid, mid, m, \mathcal{Q})$, SIM sends $(\mathsf{RETRIEVE}, sid, mid, \mathcal{R})$ to $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$. For now, the execution history for receiving (but decryption) is implicit.

Sender Dishonest. If the sender $\mathcal{S}$ is dishonest, "$\mathcal{R}$" receives a message m directly from $\mathcal{A}$. If m is not of the form (sid, mid, c, σ), the message is simply ignored. Otherwise, "$\mathcal{R}$" performs its computations according to the real protocol, i.e., it asks "$\mathcal{F}_{\mathsf{CA}}$", checks the well-formedness of the public key, and if the signature is valid. If there is an entry $(\mathsf{enc\text{-}rec}, sid', \mathsf{pk}_{\mathsf{ENC}}{}', m', \ell, c)$, with $sid' = (\mathcal{R}, sid'')$, let $m \leftarrow m'$, and otherwise $m \leftarrow \perp$. In other words, "$\mathcal{R}$" either ignores the input, "hangs", or outputs $(\mathsf{RETRIEVE}, sid, mid, m')$. In the first case, SIM simply ignores the input as well. In the second case, the execution history is still implicit if $\mathcal{R}$ becomes corrupted at this point. In the third case, check whether there is a record $(\mathsf{SEND}, \mathcal{S}, m'', r'_{\mathsf{enc}}, r'_{\mathsf{sign}}, mid, \sigma', c', \perp)$ with $m'' \neq m'$. If so, SIM updates the record $(\mathsf{SEND}, \mathcal{S}, m'', r_{\mathsf{enc}}, r_{\mathsf{sign}}, mid, \sigma, c, \perp)$ to $(\mathsf{SEND}, \mathcal{S}, m'', r'_{\mathsf{enc}}, r'_{\mathsf{sign}}, mid, \sigma', c', \mathsf{done})$ and then sends message $(\mathsf{CORRUPT}, sid, mid, \mathcal{S}, m')$ to $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$. Else, i.e., there is no record $(\mathsf{SEND}, \mathcal{S}, m, r_{\mathsf{sign}}, mid, \sigma, c, \perp)$, SIM sends $(\mathsf{SEND}, sid, mid, m')$ to $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$. $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$ directly returns control to SIM. Finally, SIM sends $(\mathsf{RETRIEVE}, sid, mid, \mathcal{R})$ to $\mathcal{F}_{\mathsf{SMT}}^{\mathcal{L}}$.

Corruption. Corruption is clearly the most interesting part of the simulation. SIM does not consider the lists (KEYGEN, sid) and $(\mathsf{KEYCONF}, sid, \mathsf{pk})$ for "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$", as these are generated honestly and thus are implicit and only blow up decryption. Moreover, if the holder of a secret key becomes corrupted, SIM does not require the lists $(\mathsf{ENCRYPT}, sid, \mathsf{pk}, m, \perp)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \perp, \mathsf{pk})$, as they are never called and are thus empty. The same is true for the decryption; a party not holding the correct secret key never calls $(\mathsf{DECRYPT}, sid, c, \perp)$, and thus never sees any $(\mathsf{PLAINTEXT}, sid, c, m, \perp)$.

The strategy is simple: for every sent message from that party, SIM needs to construct the lists $(\mathsf{ENCRYPT}, sid, \mathsf{pk}, m, \perp)$, $(\mathsf{CIPHERTEXT}, sid, c, m, \perp, \mathsf{pk})$. The randomness used for signature generation and signature key generation is already known by SIM itself. Then, SIM has to program all ciphertexts which can now be decrypted. For sent messages, this is easy. For received messages, however, this is not trivial, as SIM needs to consider hold-back messages well. Then, SIM can construct the lists $(\mathsf{DECRYPT}, sid, c, \perp)$, and $(\mathsf{PLAINTEXT}, sid, c, m, \perp)$, and can provide them to the corresponding simulator of "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$", i.e., the hybrid adversary. Note, SIM no longer needs to consider any "hanging" parties, as the execution history is implicit.

Both Parties were Honest. Upon corruption of a party $\mathcal{P}'$, the simulator receives all prior input, and output of $\mathcal{P}'$. Thus, the simulator receives the lists (INIT, sid), $(\mathsf{SEND}, sid, mid, m)$, and $(\mathsf{RETRIEVE}, sid, mid, m)$.

First, SIM needs to provide the randomness for encryptions. SIM is able to generate the lists $(\mathsf{ENCRYPT}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}'}, m, \perp)$, and also $(\mathsf{CIPHERTEXT}, sid, c, m, \perp, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}'})$ as it

has the list $(\mathsf{SEND}, \mathcal{S}, \perp_m, r_{\mathsf{sign}}, mid, \sigma, c, \perp)$, and $(\mathsf{SEND}, sid, mid, m)$. These are simply given to "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" with the *sid* corresponding to the other party. This accounts for the randomness used for encryptions. The key generation interfaces were honestly used, while the decryption interfaces were never used.

Second, SIM has to create the lists $(\mathsf{DECRYPT}, sid, c, \perp)$, and $(\mathsf{PLAINTEXT}, sid, c, m, \perp)$ for the other "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" with the *sid* corresponding to $\mathcal{P}'' \neq \mathcal{P}'$. SIM starts with completing the execution history for the received messages, which is trivial due to the records of the form $(\mathsf{SEND}, \mathcal{S}, m'', r'_{\mathsf{enc}}, r'_{\mathsf{sign}}, mid, \sigma', c', \mathsf{done})$. Next, SIM needs to consider the hold-back messages as well. Namely, SIM has still records of the form $(\mathsf{SEND}, \mathcal{P}'', \perp_m, r_{\mathsf{sign}}, mid, \sigma, c, \perp)$, where $\mathcal{P}'' \neq \mathcal{P}'$. For all these entries, SIM sends $(\mathsf{RETRIEVE}, sid, mid, \mathcal{P}'')$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, receiving $(\mathsf{RETRIEVE}, sid, mid, m, \mathcal{P}'')$ to learn each m. Thus, each record can now be updated to $(\mathsf{SEND}, \mathcal{P}'', m, r_{\mathsf{sign}}, mid, \sigma, \mathsf{true})$. Hence, SIM now knows all sent messages. SIM can now derive the records $(\mathsf{PLAINTEXT}, sid, c, m, \perp)$, which in turn, means that SIM can also complete the records of the form $(\mathsf{DECRYPT}, sid, c, \perp)$. These are then given to "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" (with the *sid* corresponding to $\mathcal{P}'$), which can then present the randomness used for the secret key. Finally, SIM has to give the randomness used for the signatures and the signature private key. As these are stored in the records of the form $(\mathsf{SEND}, \mathcal{P}', m, r_{\mathsf{enc}}, r_{\mathsf{sign}}, mid, \sigma, c, \cdot)$ and $(\mathsf{key\text{-}rec}, \mathcal{P}', \mathsf{pk}_{\mathsf{Sig},\mathcal{P}'}, \mathsf{sk}_{\mathsf{Sig},\mathcal{P}'}, r_{\mathsf{sig},\mathcal{P}'}, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}'})$, SIM can easily do so. This step also generates the execution history for decryptions.

Already a Corrupt Party. If the last party $\mathcal{P}'$ becomes corrupted, SIM needs to construct the lists $(\mathsf{ENCRYPT}, sid, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}''}, m, \perp)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \perp, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}''})$, where $\mathcal{P}''$ is the party already corrupted. As SIM has the records $(\mathsf{SEND}, \mathcal{P}'', m, r_{\mathsf{sign}}, mid, \sigma, c, \mathsf{done})$, this is straightforward. These are simply given to "$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$" with the *sid* corresponding to the already corrupt party. Finally, SIM have to construct the lists $(\mathsf{DECRYPT}, sid, c, \perp)$, and $(\mathsf{PLAINTEXT}, sid, c, m, \perp)$. As SIM has the records $(\mathsf{SEND}, \mathcal{P}'', m, r_{\mathsf{sign}}, mid, \cdot, c, \mathsf{done})$, this is easy. Finally, SIM hands over the randomness stored in records $(\mathsf{SEND}, \mathcal{P}'', \cdot, r_{\mathsf{sign}}, \cdot, \cdot, \cdot, \cdot)$ and r_{sig} stored in record $(\mathsf{key\text{-}rec}, \mathcal{P}'', \mathsf{pk}_{\mathsf{Sig},\mathcal{P}''}, \mathsf{sk}_{\mathsf{Sig},\mathcal{P}''}, r_{\mathsf{sig},\mathcal{P}''}, \mathsf{pk}_{\mathsf{ENC},\mathcal{P}''})$. The decryption history is part of "$\mathcal{P}'$".

This completes the proof.

5.6 Universally Composable Signcryption

The section illustrates the use of the non-committing encryption functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ by showing that the generic encrypt-then-sign approach that is known to be secure for building signcryption schemes under game-based definitions [Zhe97, ADR02] also works in the UC setting against adaptive adversaries, if one relies on strongly unforgeable signatures. Gjøsteen and Kråkmo [GK07] already considered UC-secure signcryption, but explicitly left security against adaptive corruptions as an open problem.

In a nutshell, signcryption is a primitive allowing a sender $\mathcal{S}$ to send an encrypted and authenticated message to a receiver $\mathcal{R}$, where both sender and receiver have their own key pairs. The goal of signcryption is to obtain better efficiency than a generic composition of encryption and signatures, but proving the generic construction secure is important to set a benchmark

1. `Key Generation.` Upon input $(\mathsf{SCKEYGEN}, sid)$ from a party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, \cdot, sid')$ and $sid \neq (\cdot, \mathcal{P}, sid')$, ignore.
   - If there is a record $(\mathsf{key\text{-}rec}, sid, \mathcal{P}, \cdot)$, ignore.
   - Send $(\mathsf{SCKEYGEN}, sid, \mathcal{P})$ to $\mathcal{A}$ and wait for $(\mathsf{SCKEYGENCONF}, sid, \mathcal{P}, v)$ from $\mathcal{A}$.
   - If $v = \perp$, ignore.
   - Create record $(\mathsf{key\text{-}rec}, sid, \mathcal{P}, v)$.
   - Output $(\mathsf{SCKEYGENCONF}, sid, v)$ to $\mathcal{P}$.
2. `SignCryption.` On input $(\mathsf{SIGNCRYPT}, sid, m, \ell, \mathsf{pk}_r)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, \mathcal{Q}, sid')$, or $\mathsf{pk}_r = \perp$, ignore.
   - If there is no record $(\mathsf{key\text{-}rec}, sid, \mathcal{P}, \mathsf{pk}_s)$, ignore.
   - If there is a record $(\mathsf{key\text{-}rec}, sid, \mathcal{Q}, v)$, let $\mathsf{pk}'_r \leftarrow v$.
   - If $\mathsf{pk}_r = \mathsf{pk}'_r$, and $\mathcal{Q}$ is honest, send $(\mathsf{SENDL}, sid, \mathcal{L}(m), \ell, \mathsf{pk}_r)$ to $\mathcal{A}$ and wait for $(\mathsf{SIGNCRYPTRES}, sid, \ell, s)$ from $\mathcal{A}$.
   - Else, send $(\mathsf{SENDM}, sid, m, \ell, \mathsf{pk}_r)$ to $\mathcal{A}$, and wait for $(\mathsf{SIGNCRYPTRES}, sid, \ell, s)$ from $\mathcal{A}$.
   - If there is a record $(\mathsf{sc\text{-}rec}, sid, \cdot, \cdot, s, \cdot, \mathsf{pk}_s, \mathsf{pk}'_r)$, pk'_r taken from $(\mathsf{key\text{-}rec}, sid, \mathcal{Q}, \mathsf{pk}'_r)$, where $sid = (\mathcal{P}, \mathcal{Q}, sid')$, ignore.
   - If $\mathsf{pk}'_r = \mathsf{pk}_r$, create record $(\mathsf{sc\text{-}rec}, sid, m, \ell, s, \mathsf{true}, \mathsf{pk}_s, \mathsf{pk}_r)$.
   - Output $(\mathsf{SIGNCRYPTRES}, sid, m, \ell, s)$ to $\mathcal{P}$.
3. `De-signcryption.` On input $(\mathsf{DESIGNCRYPT}, sid, s, \ell, \mathsf{pk}_s)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{Q}, \mathcal{P}, sid')$, or $\mathsf{pk}_s = \perp$, ignore.
   - If there is no record $(\mathsf{key\text{-}rec}, sid, \mathcal{P}, \mathsf{pk}_r)$, ignore.
   - If there is a record $(\mathsf{sc\text{-}rec}, sid, m, \ell, s, b, \mathsf{pk}_s, \mathsf{pk}_r)$, output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, b)$ to $\mathcal{P}$.
   - If there is a record $(\mathsf{key\text{-}rec}, sid, \mathcal{Q}, \mathsf{pk}_s)$, and $\mathcal{Q}$ is honest, create record $(\mathsf{sc\text{-}rec}, sid, m, \ell, s, \mathsf{false}, \mathsf{pk}_s, \mathsf{pk}_r)$, and output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \mathsf{false})$ to $\mathcal{P}$.
   - Else, send $(\mathsf{DESIGNCRYPT}, sid, s, \ell, \mathsf{pk}_s)$ to $\mathcal{A}$ and wait for $(\mathsf{DESIGNCRYPTA}, sid, s, m, \ell, \phi)$ from $\mathcal{A}$.
   - If $\phi \notin \{\mathsf{false}, \mathsf{true}\}$, ignore.
   - If there is no record $(\mathsf{sc\text{-}rec}, sid, m, \ell, \cdot, \mathsf{true}, \mathsf{pk}'_s, \mathsf{pk}_r)$, and $\mathsf{pk}'_s = \mathsf{pk}_s$, where pk'_s is taken from $(\mathsf{key\text{-}rec}, sid, \mathcal{Q}, \mathsf{pk}'_s)$, let $\phi \leftarrow \mathsf{false}$.
   - If $\phi = \mathsf{false}$, let $m \leftarrow \perp$.
   - Create record $(\mathsf{sc\text{-}rec}, sid, m, \ell, s, \phi, \mathsf{pk}_s, \mathsf{pk}_r)$.
   - Output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \phi)$ to $\mathcal{P}$.

Figure 5.9: Ideal signcryption functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{SignCrypt}}$. $\mathcal{L}$ is a leakage function

to which direct schemes can be compared. Interactive functionalities such as secure message transmission ($\mathcal{F}_{\mathsf{SMT}}$) are clearly not well suited to build a local primitive such as signcryption, so it is a good use case for the non-interactive functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. Moreover, as the signcryption directly gives rise to a protocol implementing $\mathcal{F}_{\mathsf{SMT}}$, also this functionality cannot be instantiated in the standard model.

The signcryption functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{SignCrypt}}$ is depicted in Figure 5.9. Following Gjøsteen and Kråkmo [GK07], as well as the design choices for $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, the adversary determines the ciphertext strings. Unlike Gjøsteen and Kråkmo, however, the functionality includes labels and lets public keys be determined by the adversary, modeling the case where the public keys are not necessarily certified or registered through a PKI, which avoids implying a PKI.

Explanation. Let us give a high-level description what each interface does, i.e., a more informal explanation is given to make the functionality more readable.

1. The $\mathsf{SCKEYGEN}$ interface allows the sender and receiver to generate their public keys. The key pair of the receiver is used to encrypt the message, while that of the sender will be used to authenticate it. For simplicity, the interface was merged for both participants.
2. The $\mathsf{SIGNCRYPT}$ interface allows the sender to create the signcryption of a message m to a receiver's public key pk_r. In case the public key is not the one registered for the receiver, or the receiver is corrupted, then the adversary learns the message. For the registered receiver's public key, the signcryption ciphertexts provided by the adversary must be unique.
3. The $\mathsf{DESIGNCRYPT}$ interface allows the receiver to de-signcrypt a ciphertext. If the receiver was not initialized, this interface ignores requests. In case the signcryption was honestly generated, the functionality can directly answer the requests. In all other cases, the adversary has to determine whether the ciphertext is valid and, if so, provide the plaintext. The adversary is then committed to its decision, in the sense that future de-signcryptions of the same ciphertext will yield the same result.

Protocol Description. Now, the generic encrypt-then-sign construction for a signcryption scheme using $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$ as sub-functionalities is described.

Step 1a. Key Generation (Sender $\mathcal{S}$):

a) Upon input $(\mathsf{SCKEYGEN}, sid)$, ignore, if a record $(\mathsf{key\text{-}rec}, sid, \cdot)$ exists.
b) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.
c) Create a signature key pair by inputting $(\mathsf{KEYGEN}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}})))$ to $\mathcal{F}_{\mathsf{Sig}}$.
d) Upon obtaining $(\mathsf{KEYCONF}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}})), \mathsf{pk}_{\mathsf{Sig}})$, create a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}})$.
e) Output $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{Sig}})$.

Step 1b. Key Generation (Receiver $\mathcal{R}$):

a) Upon input $(\mathsf{SCKEYGEN}, sid)$, ignore, if a record $(\mathsf{key\text{-}rec}, sid, \cdot)$ exists.
b) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.
c) Create an encryption scheme key pair by inputting $(\mathsf{KEYGEN}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
d) Upon obtaining $(\mathsf{KEYCONF}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}_{\mathsf{ENC}})$, create a record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}})$.
e) Output $(\mathsf{KEYCONF}, sid, \mathsf{pk}_{\mathsf{ENC}})$.

Step 2. Create Signcryption (Sender $\mathcal{S}$):

a) Upon input $(\mathsf{SIGNCRYPT}, sid, m, \ell, \mathsf{pk})$, ignore, if no record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{Sig}})$ exists.
b) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$ or $\mathsf{pk} = \perp$, ignore.
c) Encrypt m under pk with label $\ell' = (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk})$, i.e., input $(\mathsf{ENCRYPT}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), \mathsf{pk}, m, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
d) Sign the resulting ciphertext together with label ℓ', i.e., upon obtaining $(\mathsf{CIPHERTEXT}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}), \mathsf{pk})$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, send $(\mathsf{SIGN}, (\mathcal{S}, sid), (c, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk})))$ to $\mathcal{F}_{\mathsf{Sig}}$.
e) Upon receiving $(\mathsf{SIGNATURE}, (\mathcal{S}, sid), (c, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk})), \sigma)$, output $(\mathsf{SIGNCRYPTRES}, sid, m, \ell, (c, \sigma))$.

Step 3. De-signcryption (Receiver $\mathcal{R}$):

a) Upon input $(\mathsf{DESIGNCRYPT}, sid, s, \ell, \mathsf{pk})$, ignore if there is no record $(\mathsf{key\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}})$ or $\mathsf{pk} = \perp$.

b) If $s \neq (c, \sigma)$ or $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.

c) Verify the signature, i.e., input $(\mathsf{VERIFY}, (\mathcal{S}, sid), (c, (\ell, \mathsf{pk}, \mathsf{pk}_{\mathsf{ENC}})), \sigma, \mathsf{pk})$ to $\mathcal{F}_{\mathsf{Sig}}$. Upon receiving $(\mathsf{VERIFY}, (\mathcal{S}, sid), (m, (\ell, \mathsf{pk}, \mathsf{pk}_{\mathsf{ENC}})), \sigma, \mathsf{pk}, v)$ from $\mathcal{F}_{\mathsf{Sig}}$, output $(\mathsf{DESIGNCRYPT}, sid, s, \perp, (\ell, \mathsf{pk}, \mathsf{pk}_{\mathsf{ENC}}), \mathsf{false})$, if $v = \mathsf{false}$. Decrypt the ciphertext, i.e., send $(\mathsf{DECRYPT}, (\mathcal{R}, sid), c, (\ell, \mathsf{pk}, \mathsf{pk}_{\mathsf{ENC}}))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. Upon receiving input $(\mathsf{PLAINTEXT}, (\mathcal{R}, sid), c, m, (\ell, \mathsf{pk}, \mathsf{pk}_{\mathsf{ENC}}))$, output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \mathsf{true})$.

Theorem 5.18. *The above construction securely realizes $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ in the $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid model with adaptive corruptions without secure erasures.*

Proof. Given a $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid-model adversary $\mathcal{A}$, construct a simulator SIM so that no environment can distinguish running with $\mathcal{A}$ and the real protocol in the $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid world from running with SIM and $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ in the ideal world. Note that, apart from interacting correctly with $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, the simulator must also play the role of the subfunctionalities $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$ to $\mathcal{A}$ when they are called by corrupt parties. The simulator SIM proceeds as follows:

Key Generation ($\mathcal{S}$). Upon input $(\mathsf{SCKEYGEN}, sid, \mathcal{S})$ from $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, send $(\mathsf{KEYGEN}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}})))$ to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}}))), \mathsf{pk}_{\mathsf{Sig}})$ from $\mathcal{A}$. Create a record $(\mathsf{key\text{-}rec}, sid, \mathcal{S}, \mathsf{pk}_{\mathsf{Sig}})$ and send $(\mathsf{SCKEYGENCONF}, sid, \mathcal{S}, \mathsf{pk}_{\mathsf{Sig}})$ to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$.

Key Generation ($\mathcal{R}$). Upon receiving $(\mathsf{SCKEYGEN}, sid, \mathcal{R})$ from $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, send $(\mathsf{KEYGEN}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}))), \mathsf{pk}_{\mathsf{ENC}})$ from $\mathcal{A}$. Create a record $(\mathsf{key\text{-}rec}, sid, \mathcal{R}, \mathsf{pk}_{\mathsf{ENC}})$ and send $(\mathsf{SCKEYGENCONF}, sid, \mathcal{R}, \mathsf{pk}_{\mathsf{ENC}})$ to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$.

Encryption. Upon receiving $(\mathsf{ENCRYPT}, sid', \mathsf{pk}'_{\mathsf{ENC}}, m, \ell)$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, asking $\mathcal{A}$ to provide a ciphertext c, creating an encryption record $(\mathsf{enc\text{-}rec}, sid, \mathsf{pk}'_{\mathsf{ENC}}, m, \ell, c)$. Afterwards, it returns, i.e., outputs $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, \mathsf{pk}'_{\mathsf{ENC}})$.

Decryption. Upon receiving $(\mathsf{DECRYPT}, sid', c, \ell)$ from $\mathcal{A}$ in name of the corrupt receiver $\mathcal{R}$, SIM executes the real code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, asking $\mathcal{A}$ for a plaintext m if necessary, creating a decryption record $(\mathsf{dec\text{-}rec}, sid, m, \ell, c)$, and returning $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

Signing. Upon receiving (SIGN, sid', m) from $\mathcal{A}$ in name of the corrupt sender $\mathcal{S}$, SIM executes the real code of $\mathcal{F}_{\mathsf{Sig}}$, asking $\mathcal{A}$ for a signature σ, creating a signing record $(\mathsf{sig\text{-}rec}, sid, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{true})$, and returning $(\mathsf{SIGNATURE}, sid, m, \sigma)$.

Verification. Upon receiving $(\mathsf{VERIFY}, sid', m, \sigma, \mathsf{pk}'_{\mathsf{Sig}})$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}_{\mathsf{Sig}}$, asking $\mathcal{A}$ for a verification outcome ϕ if necessary, creating signature record $(\mathsf{sig\text{-}rec}, sid', m, \cdot, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{true})$ or $(\mathsf{sig\text{-}rec}, sid', m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \phi)$, and returning $(\mathsf{VERIFY}, sid', m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \phi)$.

Signcryption. When $\mathcal{S}$ and $\mathcal{R}$ are both honest and $\mathsf{pk}_r = \mathsf{pk}_{\mathsf{ENC}}$, SIM is notified by $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ by receiving $(\mathsf{SENDL}, sid, \mathcal{L}(m), \ell, \mathsf{pk}_r)$. The simulator doesn't know the message m, but obtains a ciphertext by sending $(\mathsf{ENCRYPTL}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}), \mathsf{pk}_{\mathsf{ENC}}, \mathcal{L}(m), (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{ENC}}))$ to $\mathcal{A}$ and waiting for $(\mathsf{CIPHERTEXT}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}), c)$ from $\mathcal{A}$. It then creates an incomplete encryption record $(\mathsf{enc\text{-}rec}, sid, \mathsf{pk}_{\mathsf{ENC}}, \perp, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{ENC}}), c)$, indicating that it doesn't know the corresponding message m. It then obtains a signature σ for (c, ℓ) using the procedure for signing simulation above and sends $(\mathsf{SIGNCRYPTRES}, sid, (\ell, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{ENC}}), s = (c, s))$ back to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$.

When $\mathcal{S}$ is honest and ($\mathcal{R}$ is corrupt or $\mathsf{pk}_r \neq \mathsf{pk}_{\mathsf{ENC}}$), then SIM executes the real signcryption algorithm with the code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$.

Designcryption. When a new de-signcryption request occurs, SIM is notified by $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ by a message $(\mathsf{DESIGNCRYPT}, sid, s = (c, \sigma), \ell, \mathsf{pk}_s)$. First check whether the signature σ should be deemed valid for message $(c, (\ell, \mathsf{pk}_s, \mathsf{pk}_{\mathsf{ENC}}))$ by performing the verification simulation above. If not, then SIM sends $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, indicating that the signcryption is invalid. If so, then it must be the case that $\mathsf{pk}_s \neq \mathsf{pk}_{\mathsf{Sig}}$, because the strong unforgeability enforced by $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ would have rejected the ciphertext already. If c was part of a signcryption generated by the honest sender (for which the corresponding plaintext is not known), then SIM sends $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, because the ciphertext label of c includes the wrong sender's public key $\mathsf{pk}_{\mathsf{Sig}} \neq \mathsf{pk}_s$. Otherwise, $\mathcal{A}$ decrypts c by performing the decryption simulation above to obtain the plaintext message m. If $m = \perp$, then send $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$, else send $(\mathsf{DESIGNCRYPTA}, sid, s, m, \ell, \mathsf{true})$ back to $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$.

Corruption. When a party is corrupted, then SIM obtains the full input and output history of that party. Based on this history, SIM can also compile the list of inputs and outputs of that party to the $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$ sub-functionalities, which it could not do earlier because some of the messages were unknown. It submits this full list of inputs and outputs to $\mathcal{A}$. □

It is stressed that defining a functionality for multiple senders, and receivers, is straightforward.

5.7 Conclusion

This chapter defined and gave a secure realization of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, a UC functionality for local public-key encryption, in a setting of adaptive corruptions without erasures. It also introduced a new game-based notion of $\mathsf{FULL\text{-}SIM}$ security, which was shown to be equivalent to the UC definition. The non-interactive construction assumes the existence of trapdoor one-way permutations in the random-oracle model. It is very efficient, as it only has a constant overhead, only requires one evaluation of a trapdoor permutation, and a handful of hash evaluations. Due to Nielsen's impossibility result for round-optimal adaptively secure secure message transmission, a standard model instantiation of the primitive was also shown to be impossible.

Still, it is reasonable to believe that the functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and its instantiation can be a very useful building block for other UC-secure protocols, which until now had to provide direct constructions tailored to one specific application scenario. As an example, it was shown how

$\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ can be used to model and realize the signcryption functionality $\mathcal{F}_{\mathsf{SignCrypt}}^{\mathcal{L}}$ in the presence of adaptive adversaries without erasures.

Chapter 6

Chameleon-Hashes with Ephemeral Trapdoors and Their Applications to Invisible Sanitizable Signatures

The results of this chapter have already been published [CDK+17].

Abstract. A chameleon-hash function is a hash function that involves a trapdoor the knowledge of which allows one to find arbitrary collisions in the domain of the function. In this chapter, the notion of *chameleon-hash functions with ephemeral trapdoors* is introduced. Such hash functions feature additional, i.e., ephemeral, trapdoors which are chosen by the party computing a hash value. The holder of the main trapdoor is then unable to find a second pre-image of a hash value unless also provided with the ephemeral trapdoor used to compute the hash value. This chapter presents a formal security model for this new primitive as well as provably secure instantiations. The first instantiation is a generic black-box construction from any secure chameleon-hash function. This chapter further provides three direct constructions based on standard assumptions. The new primitive has some appealing use-cases, including a solution to the long-standing open problem of *invisible* sanitizable signatures, which is also presented in this chapter.

Roadmap. The problem statement is presented in Section 6.1. The new definition of chameleon-hashes with ephemeral trapdoors is given in Section 6.2, while the corresponding constructions are presented in Section 6.3. Section 6.4 contains a small note on chameleon-signatures, while Section 6.5 proposes the new primitive of invisible sanitizable signatures as the application scenario for the new chameleon-hashes. This chapter is concluded in Section 6.6.

6.1 Introduction

Chameleon-hash functions, also called trapdoor-hash functions, are hash functions that feature a trapdoor that allows one to find arbitrary collisions in the domain of the functions. However, chameleon-hash functions are collision resistant as long as the corresponding trapdoor (or secret key) is not known. More precisely, a party who is privy of the trapdoor is able to find arbitrary

collisions in the domain of the function. Example instantiations include trapdoor-commitments, and equivocal commitment schemes.

One prominent application of this primitive are chameleon signatures [KR00b]. Here, the intended recipient—who knows the trapdoor—of a signature σ for a message m can equivocate it to another message m' of his choice. This, in turn, means that a signature σ cannot be used to convince any other party of the authenticity of m, as the intended recipient could have "signed" arbitrary messages on its own. Many other applications appear in the literature, some of which is discussed in the related work section. However, all current constructions are "all-or-nothing" in that a party who computes a hash with respect to some public key cannot prevent the trapdoor holder from finding collisions. This can be too limiting for some use-cases.

6.1.1 Contribution

This chapter introduces a new primitive dubbed chameleon-hash functions with ephemeral trapdoors. In a nutshell, this primitive requires that a collision in the hash function can be computed only when two secrets are known, i.e., the main trapdoor, and an ephemeral one. The main trapdoor is the secret key corresponding to the chameleon-hash function public key, while the second, ephemeral, trapdoor is generated by the party computing the hash value. The latter party can then decide whether the holder of the long-term secret key shall be able to equivocate the hash by providing or withholding the second trapdoor information. Also a corresponding formal security model for this new primitive is presented. This chapter also includes stronger definitions for existing chameleon-hash functions not considered before. Thus includes the new notion of uniqueness and fully adaptive security. These new notions may also be useful in other scenarios.

Moreover, four provably secure constructions for chameleon-hash functions with ephemeral trapdoors are presented. The first is bootstrapped, while the three direct constructions are built on RSA-like and the DL assumption. The new primitive has some interesting applications, including the first provably secure instantiation of *invisible* sanitizable signatures, which are also presented. Additional applications of this new primitive may include revocable signatures [HKY15], but also simulatable equivocable commitments [Fis01]. However, in contrast to equivocable commitments, it is wanted that parties can actually equivocate, not only a simulator. Therefore, it was chosen to call this primitive a chameleon-hash function rather than a commitment. Note, the primitive is different from "double-trapdoor chameleon-hash functions" [BCG07, CRFG08, LZCS16], where knowing one out of two secrets is enough to produce collisions.

6.1.2 Related Work

Standard chameleon-hashes were introduced by Krawczyk and Rabin [KR00b], based on the work done by Brassard et al. [BCC88]. Later, they have been ported to the identity-based setting, i.e., ID-based chameleon-hash functions, where the holder of some master secret key can extract new secret keys for some identity [AdM04, BDD+11, RMS08, ZSS03]. However, most of the schemes presented suffer from the key-exposure problem [AdM04, KR00b]. Key exposure means that seeing a single collision in the hash allows to find further collisions by extracting the

corresponding trapdoor, i.e., the secret key.[1] This problem was addressed by the introduction of "key-exposure free" chameleon-hashes [AdM04,CTZD10,CZK04,GLW09,GWX07,RMS08], which prohibit extracting the secret key if a collision was seen. This also includes combinations of both techniques [CZS+10]. This allows re-using the secret, which is also the goal of the presented primitive. However, the definition of collision-resistance is defined w.r.t. some additional label L. Moreover, they do *not* prohibit that once a collision is made public for a label L, an adversary can produce additional collisions for the given hash w.r.t. that label L. All mentioned approaches are thus orthogonal to the goal here, as in the discussed case two keys at *the same time* are required. Brzuska et al. then proposed a formal framework for tag-based chameleon-hashes secure under random-tagging attacks, i.e., they add an additional tag to the input to the hashing algorithm [BFF+09].

Additional related work is discussed when presenting the applications of the new primitive.

6.1.3 Chameleon-Hashes

Now, a "standard" chameleon-hash is defined. The framework is based upon the work done by Ateniese et al. and Brzuska et al. [AMVA17, BFF+09], but adapted to fit the given notation. Additionally, some extended security definitions are provided.

Definition 6.1. *A chameleon-hash* CH *consists of five algorithms* $(\mathsf{PPGen}_{\mathsf{CH}}, \mathsf{KeyGen}_{\mathsf{CH}}, \mathsf{Hash}_{\mathsf{CH}}, \mathsf{Check}_{\mathsf{CH}}, \mathsf{Adapt}_{\mathsf{CH}})$*, such that:*

$\mathsf{PPGen}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CH}}$ *on input security parameter* λ *outputs public parameters of the scheme:*

$$\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$$

It is assumed that $\mathsf{pp}_{\mathsf{CH}}$ *is implicit input to all other algorithms.*

$\mathsf{KeyGen}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{CH}}$ *given the public parameters* $\mathsf{pp}_{\mathsf{CH}}$ *outputs the private and public keys of the scheme:*

$$(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$$

$\mathsf{Hash}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{Hash}_{\mathsf{CH}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{CH}}$*, and a message* m *to hash. It outputs a hash* h*, and some randomness* r*:*

$$(h, r) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$$

The randomness r *is also sometimes called "check value" [AMVA17].*

$\mathsf{Check}_{\mathsf{CH}}$**.** *The deterministic algorithm* $\mathsf{Check}_{\mathsf{CH}}$ *gets as input the public key* $\mathsf{pk}_{\mathsf{CH}}$*, a message* m*, randomness* r*, and a hash* h*. It outputs a decision* $d \in \{\mathsf{false}, \mathsf{true}\}$ *indicating whether the hash* h *is valid:*

$$d \leftarrow \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h)$$

$\mathsf{Adapt}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{Adapt}_{\mathsf{CH}}$ *on input of secret key* $\mathsf{sk}_{\mathsf{CH}}$*, the old message* m*, the old randomness* r*, hash* h*, and a new message* m' *outputs new randomness* r'*:*

$$r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$$

[1] In the case of identity-based chameleon-hashes w.r.t. to some identity.

Experiment $\mathsf{Indistinguishability}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$
$b \xleftarrow{\$} \{0,1\}$
$a \xleftarrow{\$} \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,b),\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CH}})$
where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{CH}}, m, m'$ and b:
$(h, r) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m')$
$(h', r') \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$
$r'' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r', h')$
If $r = \bot \ \vee \ r'' = \bot$, return $\bot$
if $b = 0$:
return (h, r)
if $b = 1$:
return (h', r'')
return 1, if $a = b$
return 0

Figure 6.1: CH Indistinguishability

Correctness. Each CH must have a correctness property. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$, for all $m \in \mathcal{MS}$, for all $(h, r) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$, for all $m' \in \mathcal{MS}$, it holds for all $r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$, that $\mathsf{true} = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m', r', h)$ holds. This definition captures perfect correctness. The randomness is drawn by $\mathsf{Hash}_{\mathsf{CH}}$, and not outside. This was done to capture "private-coin" constructions [AMVA17].

Indistinguishability. Indistinguishability requires that the randomnesses r does not reveal if it was obtained through $\mathsf{Hash}_{\mathsf{CH}}$ or $\mathsf{Adapt}_{\mathsf{CH}}$. The messages are chosen by the adversary. The perfect indistinguishability definition of Brzuska et al. [BFF+09] is relaxed to a computational version, which is enough for most use-cases, including the one presented.

Note, the experiments returns $\bot$ in some cases in the $\mathsf{HashOrAdapt}$ oracle, as the adversary may try to enter a message $m \notin \mathcal{MS}$, even if $\mathcal{MS} = \{0,1\}^*$, which makes the algorithm output $\bot$. If one would not do this, the adversary could trivially decide indistinguishability. For similar reasons these checks are also included in other definitions.

Definition 6.2 (Indistinguishability)**.** *A chameleon-hash* CH *is* ***indistinguishable****, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{Indistinguishability}^{\mathsf{CH}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.1.

Experiment $\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, r^*, m'^*, r'^*, h^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CH}})$
where oracle $\mathsf{Adapt}'_{\mathsf{CH}}$ on input $\mathsf{sk}_{\mathsf{CH}}, m, m', r, h$:
Return $\perp$, if $\mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) \neq \mathsf{true}$
$r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$
If $r' = \perp$, return $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
return r'
return 1, if $\mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m^*, r^*, h^*) = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m'^*, r'^*, h^*) = \mathsf{true} \wedge$
$m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
return 0

Figure 6.2: CH Collision Resistance

Collision Resistance. Collision resistance says, that even if an adversary has access to an adapt oracle, it cannot find any collisions for messages other than the ones queried to the adapt oracle. Note, this is an even stronger definition than key-exposure freeness [AdM04]: key-exposure freeness only requires that one cannot find a collision for some new "tag", i.e., for some auxiliary value for which the adversary has never seen a collision.

Definition 6.3 (Collision-Resistance)**.** *A chameleon-hash* CH *is collision-resistant, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.2.

Uniqueness. Uniqueness requires that it is hard to come up with two different randomness values for the same message m^* such that the hashes are equal, for the same adversarially chosen pk^*.

Definition 6.4 (Uniqueness)**.** *A chameleon-hash* CH *is unique, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.3.

Definition 6.5 (Secure Chameleon-Hashes)**.** *A chameleon-hash* CH *is said to be secure, if it is correct, indistinguishable, and collision-resistant.*

Experiment $\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$
$(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\mathsf{CH}})$
return 1, if $\mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathsf{true}$
$\wedge\ r^* \neq r'^*$
return 0

Figure 6.3: CH Uniqueness

Uniqueness is not considered as a fundamental security property, as it depends on the concrete use-case whether this notion is required.

Subsequently is shown how to construct such a chameleon-hash, which is essentially a modified construction of the one given by Brzuska et al. [BFF+09].

The Modified Construction. Now the modified construction is presented, inspired by the ideas given by Brzuska et al. [BFF+09]. It should be clear that indistinguishability still holds in this setting, except that the modified construction achieves stronger collision-resistance under a different assumption, and is unique, which is proven on its own.

Construction 6.6 (Modified Chameleon-Hash)**.** *Let $\mathcal{H}_N : \{0,1\}^* \to \mathbb{Z}_N^*$, $N \in \mathbb{N}$, denote a random oracle. Now, let* $\mathsf{CH} := (\mathsf{PPGen}_{\mathsf{CH}}, \mathsf{KeyGen}_{\mathsf{CH}}, \mathsf{Hash}_{\mathsf{CH}}, \mathsf{Check}_{\mathsf{CH}}, \mathsf{Adapt}_{\mathsf{CH}})$ *such that:*

$\mathsf{PPGen}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CH}}$ *generates the public parameters in the following way:*

1. *Call* RSAGen *with the restriction $e > N$, and e prime. Return e.*

$\mathsf{KeyGen}_{\mathsf{CH}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{CH}}$ *generates the key pair in the following way:*

1. *Generate p, q using* $\mathsf{RSAGen}(1^\lambda)$.
2. *Let $N = pq$.*
3. *Compute d such that $ed \equiv 1 \bmod \varphi(N)$.*
4. *Return* $(\mathsf{pk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{CH}}) = (N, d)$.

$\mathsf{Hash}_{\mathsf{CH}}$**.** *To hash a message m w.r.t.* $\mathsf{pk}_{\mathsf{CH}}$ *do:*

1. *Draw* $r \xleftarrow{\$} \mathbb{Z}_N^*$.
2. *Let* $h \leftarrow \mathcal{H}_N(m) r^e \bmod N$.
3. *Return* (h, r).

$\mathsf{Check}_{\mathsf{CH}}$**.** *To check a hash h' w.r.t. a message m, randomness r, and* $\mathsf{pk}_{\mathsf{CH}}$ *do:*

1. *If* $r \notin \mathbb{Z}_N^*$, *return* false.
2. *Let* $h \leftarrow \mathcal{H}_N(m) r^e \bmod N$.

3. *Return* true*, if* $h = h'$*, and* false *otherwise.*

$\mathsf{Adapt}_{\mathsf{CH}}$**.** *To find a collision w.r.t.* m*,* m'*, randomness* r*, hash* h*, and* $\mathsf{sk}_{\mathsf{CH}}$ *do:*

1. *If* $\mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) = \mathsf{false}$*, return* $\perp$*.*
2. *If* $m = m'$*, return* r*.*
3. *Let* $g \leftarrow \mathcal{H}_N(m)$*, and* $y \leftarrow gr^e \bmod N$*.*
4. *Let* $g' \leftarrow \mathcal{H}_N(m')$*.*
5. *Return* $r' \leftarrow (y(g'^{-1}))^d \bmod N$*.*

Theorem 6.7. *If the one-more RSA-inversion assumption holds, then the above construction is secure in the random oracle model.*

For the proofs of the given constructions, two auxiliary lemmas which are proven subsequently are needed. The first one is well-known, while the second one allows an approximation for one of the reductions to succeed.

Lemma 6.8. *Let* $N \geq 2$ *be any arbitrary integer, and* $e > N$ *be any prime. Then,* $r^e \equiv r'^e \bmod N$ *implies* $r = r'$*, if* $r \in \mathbb{Z}_N^*$ *and* $r' \in \mathbb{Z}_N^*$*.*

Proof. Because of $e > N \geq \varphi(N)$ and e prime it holds that $\gcd(e, \varphi(N)) = 1$. Therefore, there exists d such that $de \equiv 1 \bmod \varphi(N)$. It holds now that $r^{ed} \equiv r'^{ed} \bmod N$, and the claim follows as $x^{\varphi(N)} \equiv 1 \bmod N$ for all $x \in \mathbb{Z}_N^*$. □

Lemma 6.9. *There exists a polynomial* $p(\cdot)$ *such that for every adversary* $\mathcal{A}$ *that on input* N *outputs* N' *of the same bit-length, it holds that:*

$$\Pr\left[y \in \mathbb{Z}_{NN'}^* : (N, p, q, e, d) \xleftarrow{\$} \mathsf{RSAGen}(1^\lambda), N' \xleftarrow{\$} \mathcal{A}(N), y \xleftarrow{\$} \mathbb{Z}_N^*\right] > \frac{1}{p(\cdot)}$$

Thus, the given probability is non-negligible.

Before proving the lemma, recap the following fact:

Fact 6.10 (Th. 8.8.7 [BS96])**.** *Let* $n \geq 3$ *be an integer. It then holds that:*

$$\frac{\varphi(N)}{N} \geq \frac{1}{e^\gamma \log\log N + \frac{3}{\log\log N}},$$

where $\gamma = 0.57721\ldots$ *is the Euler-Mascheroni constant.*

Now note that this ratio is noticeable in λ for every number N output by an efficient algorithm on input 1^λ, as the bit-length of such an n is polynomially bounded in λ, say by $p(\cdot)$, and thus $N \leq 2^{p(\lambda)}$. Thus, the denominator of the above ratio is bounded by $O(\log(p(\lambda)))$ and thus by $O(\lambda)$.

Proof. First note that $y \in \mathbb{Z}_{NN'}^*$ if and only if $y \in \mathbb{Z}_N^*$ and $y \in \mathbb{Z}_{N'}^*$. The former holds true by definition; one can thus replace $\mathbb{Z}_{NN'}^*$ by $\mathbb{Z}_{N'}^*$ in the definition.

If $N = N'$ the sought probability is 1. Otherwise, note that replacing $\mathbb{Z}_N^*$ by $\mathbb{Z}_N$ only changes the probability by a negligible amount, as for an RSA modulus $\varphi(N)/N$ is overwhelming. In

the following the probability space (i.e., $(N, p, q, e, d) \leftarrow \mathsf{RSAGen}(1^\lambda), N' \xleftarrow{\$} \mathcal{A}(N), y \xleftarrow{\$} \mathbb{Z}_N$) is denoted by Ω. Furthermore, $\Omega[\![N']\!]$ denotes the same probability space except that $y \leftarrow \mathbb{Z}_N$ is replaced by $y \leftarrow \mathbb{Z}_{N'}$.

Let $N > N'$. Then follows:

$$\begin{aligned}
\Pr[y \in \mathbb{Z}^*_{N'} : \Omega] &= \Pr[y \in \mathbb{Z}^*_{N'} \wedge 0 \leq y < N' : \Omega] + \Pr[y \in \mathbb{Z}^*_{N'} \wedge N' \leq y < N : \Omega] \\
&\geq \Pr[y \in \mathbb{Z}^*_{N'} \wedge 0 \leq y < N' : \Omega] \\
&= \Pr[y \in \mathbb{Z}^*_{N'} | 0 \leq y < N' : \Omega] \cdot \Pr[0 \leq y < N' : \Omega] \\
&= \Pr[y \in \mathbb{Z}^*_{N'} : \Omega[\![N']\!]] \cdot \frac{N'}{N} \\
&\geq \frac{\varphi(N')}{N'} \cdot \frac{1}{2} = \frac{1}{O(\lambda)}.
\end{aligned}$$

For $N < N'$ it holds that:

$$\begin{aligned}
\Pr[y \in \mathbb{Z}^*_{N'} : \Omega] &= \Pr[y \in \mathbb{Z}^*_{N'} | 0 \leq y < N : \Omega[\![n']\!]] \\
&= \Pr[y \in \mathbb{Z}^*_{N'} \wedge 0 \leq y < N : \Omega[\![N']\!]] \cdot \Pr[0 \leq y < N : \Omega[\![N']\!]] \\
&\geq \Pr[y \in \mathbb{Z}^*_{N'} : \Omega[\![N']\!]] \cdot \frac{N}{N'} \\
&\geq \frac{\varphi(N')}{N'} \cdot \frac{1}{2} = \frac{1}{O(\lambda)}
\end{aligned}$$

The claim of the lemma follows. □

Now, the proof of the first construction is given.

Proof. Each property is proven separately.

Indistinguishability. Indistinguishability is straightforward to see; the randomness r is freshly drawn in the challenge oracle and RSA defines a permutation. The proof is therefore omitted.

Collision-Resistance. Now is proven that the above construction is collision-resistant.

Game 0: This is the original collision-resistance game.

Game 1: As Game 0, but instead of using the e from the system parameters, e received from a one-more RSA challenger is embedded as $\mathsf{PPGen}_{\mathsf{CH}}$. Note, one can still determine d—and therefore honestly simulate all oracles—as the n from the challenger is not used at this point and can thus freely choose it. This does not change the view of the adversary, as the received e is distributed identically to the prior game.

Game 2: As Game 1, but abort, if the adversary was able to generate a collision $(m^*, r^*, m'^*, r'^*, h^*)$. Let this event be denoted E_2. Assume that event E_2 does happen with non-negligible probability. One can then build an adversary $\mathcal{B}$ which breaks the one-more RSA-inversion assumption. Without loss of generality, assume that the adversary makes all the random oracle queries before outputting the messages (otherwise, $\mathcal{B}$ does them). The adversary $\mathcal{B}$ proceeds as follows. In the first step, the challenge N_c is embedded in $\mathsf{pk}_{\mathsf{CH}}$. Clearly, as the distributions are the same, this is only a conceptual change so far. In the second step, for each *new* random oracle query m_i, $\mathcal{B}$ asks its challenge oracle $\mathcal{C}$ to provide a challenge $c_i \in \mathbb{Z}_N^*$. This challenge is embedded as the response to m_i. It stores $(m_i, c_i, \perp)$ in an internal table. This does not change the view of the adversary either. However, it remains open how to simulate the adaption oracle. Assume, for now, that m_0 is supposed to be adapted to m_1. If $m_0 = m_1$, proceed as in the algorithm. If there is no tuple (m_0, c_0, z_0), with $z_0 \neq \perp$, query the inversion oracle with c_0 to receive z_0. Update record $(m_0, c_0, \perp)$ to (m_0, c_0, z_0). If there is no tuple (m_1, c_1, z_1), with $z_1 \neq \perp$, query the inversion oracle with c_1 to receive z_1. Update record $(m_1, c_1, \perp)$ to (m_1, c_1, z_1). Return $r' \leftarrow r z_0 (z_1)^{-1} \bmod N$. Then, at some point in time, the adversary returns (m, r, m', r', h). One then knows (by construction) that $\mathcal{H}_N(m) r^e \equiv \mathcal{H}_N(m') r'^e \bmod N$. If there is no record for m *and* no record for m', query the inversion oracle for the root z of $\mathcal{H}_N(m)$, and update record $(m, c, \perp)$ to (m, c, z). Then, by definition of the security game, one knows that m' is fresh, and there exists a record $(m', c', \perp)$. One can then extract $\mathcal{H}_N(m')^d = z'$ by calculating $\mathcal{H}_N(m')^d \equiv r'^{-1} z r \bmod N$. As therefore the adversary $\mathcal{A}$ has inverted more challenges than the inversion oracle was queried, $\mathcal{B}$ can return the list $\{(c_i, z_i)\}$ for each entry where (m_i, c_i, z_i), $z_i \neq \perp$ exists, along with $(c', \mathcal{H}_N(m')^d)$. As now the adversary has no other way to win its game, collision-resistance is proven, as each hop only changes the view of the adversary negligibly.

Uniqueness. Now is proven that the above construction is unique using a sequence of games:

Game 0: The original uniqueness game.

Game 1: As Game 0, but the challenger aborts, if the adversary finds randomness $r^* \neq r'^*$, a public key $\mathsf{pk}^* = N$, a message m^*, and a hash h^* such that $\mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathsf{true}$. Let this abort event be denoted E_1. This cannot happen, as RSA (with the given restrictions on e and r) defines a permutation, and random oracles behave as functions, regardless of the choice of N. See also Lemma 6.8. This proves that the construction is unique.

As now collision-resistance and uniqueness is proven, security of the construction is finally proven. □

6.2 Chameleon-Hashes with Ephemeral Trapdoors

As already mentioned, a chameleon-hash with ephemeral trapdoor (CHET) allows to prevent the holder of the trapdoor $\mathsf{sk}_{\mathsf{CHET}}$ from finding collisions, as long as no additional ephemeral trapdoor sktd is known. This additional ephemeral trapdoor is chosen freshly for each new hash, and providing, or withholding, this trapdoor thus allows to decide upon each hash computation

if finding a collision is possible for the holder of the long-term trapdoor. Hence, one needs to introduce a new framework given next, which is also accompanied by suitable security definitions.

Definition 6.11 (Chameleon-Hashes with Ephemeral Trapdoors)**.** *A chameleon-hash with ephemeral trapdoors* CHET *is a tuple of five algorithms* $(\mathsf{PPGen}_{\mathsf{CHET}}, \mathsf{KeyGen}_{\mathsf{CHET}}, \mathsf{Hash}_{\mathsf{CHET}}, \mathsf{CheckHash}_{\mathsf{CHET}}, \mathsf{Adapt}_{\mathsf{CHET}})$*, such that:*

$\mathsf{PPGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CHET}}$ *outputs the public parameters:*

$$\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$$

For simplicity, it is assumed that $\mathsf{pp}_{\mathsf{CHET}}$ *is an implicit input to all other algorithms.*

$\mathsf{KeyGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{CHET}}$ *given the public parameters* $\mathsf{PPGen}_{\mathsf{CHET}}$ *outputs the long-term private and public keys of the scheme:*

$$(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$$

$\mathsf{Hash}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{Hash}_{\mathsf{CHET}}$ *gets as input the public key of the scheme* $\mathsf{pk}_{\mathsf{CHET}}$*, and a message* m *to hash. It outputs a hash* h*, randomness* r*, and the ephemeral trapdoor information* sktd*:*

$$(h, r, \mathsf{sktd}) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$$

$\mathsf{CheckHash}_{\mathsf{CHET}}$**.** *The deterministic algorithm* $\mathsf{CheckHash}_{\mathsf{CHET}}$ *gets as input the public key of the scheme* $\mathsf{pk}_{\mathsf{CHET}}$*, a message* m*, a hash* h*, and randomness* r*. It outputs a decision bit* $d \in \{\mathsf{false}, \mathsf{true}\}$*, indicating whether the given hash is correct w.r.t. to the input values:*

$$d \leftarrow \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, r, h)$$

$\mathsf{Adapt}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{Adapt}_{\mathsf{CHET}}$ *gets as input* $\mathsf{sk}_{\mathsf{CHET}}$*, the old message* m*, the old randomness* r*, the new message* m'*, the hash* h*, and the trapdoor information* sktd *and outputs new randomness* r'*:*

$$r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, m, m', r, h, \mathsf{sktd})$$

Correctness. For each CHET the usual correctness properties must hold. In particular, it is required that for all security parameters $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$, for all $m \in \mathcal{MS}$, for all $(h, r, \mathsf{sktd}) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$, $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, r, h) = \mathsf{true}$ holds, and additionally for all $m' \in \mathcal{MS}$, for all $r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, m, m', r, h, \mathsf{sktd})$, it holds that $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m', r', h) = \mathsf{true}$. This definition captures perfect correctness. Also, some security guarantees are required, which are introduced next.

Experiment $\mathsf{Indistinguishability}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$
$b \xleftarrow{\$} \{0,1\}$
$a \xleftarrow{\$} \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,b),\mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CHET}})$
where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{CHET}}, m, m'$ and b:
let $(h, r, \mathsf{sktd}) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m')$
let $(h', r', \mathsf{sktd}') \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$
let $r'' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, m, m', r', h', \mathsf{sktd}')$
if $r'' = \bot \ \vee \ r' = \bot$, return $\bot$
if $b = 0$:
return (h, r, sktd)
if $b = 1$:
return $(h', r'', \mathsf{sktd}')$
return 1, if $a = b$
return 0

Figure 6.4: CHET Indistinguishability

Indistinguishability. Indistinguishability requires that the randomnesses r does not reveal if it was obtained through $\mathsf{Hash}_{\mathsf{CHET}}$ or $\mathsf{Adapt}_{\mathsf{CHET}}$. In other words, an outsider cannot decide whether a message is the original one or not.

Definition 6.12 (Indistinguishability)**.** *A chameleon-hash with ephemeral trapdoor* CHET *is indistinguishable, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{Indistinguishability}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.4.

Public Collision Resistance. Public collision resistance requires that, even if an adversary has access to an $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle, it cannot find any collisions by itself. Clearly, the collision must be fresh, i.e., must not be produced using the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle to avoid trivial wins of the adversary.

Definition 6.13 (Public Collision-Resistance)**.** *A chameleon-hash with ephemeral trapdoor* CHET *is publicly collision-resistant, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{PublicCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.5.

Experiment $\mathsf{PublicCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, r^*, m'^*, r'^*, h^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CHET}})$
where oracle $\mathsf{Adapt}'_{\mathsf{CHET}}$ on input $\mathsf{sk}_{\mathsf{CHET}}, m, m', r, \mathsf{sktd}$ and h:
return $\perp$, if $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, r, h) = \mathsf{false}$
$r' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, m, m', r, h, \mathsf{sktd})$
If $r' = \perp$, return $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
return r'
return 1, if $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m^*, r^*, h^*) = \mathsf{true} \wedge$
$\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m'^*, r'^*, h^*) = \mathsf{true} \wedge$
$m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
return 0

Figure 6.5: CHET Public Collision-Resistance

Experiment $\mathsf{PrivateCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(\mathsf{pk}^*, \mathsf{state}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\mathsf{CHET}})$
$(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Hash}'_{\mathsf{CHET}}(\mathsf{pk}^*,\cdot)}(\mathsf{state})$
where oracle $\mathsf{Hash}'_{\mathsf{CHET}}$ on input pk^* and m:
$(h, r, \mathsf{sktd}) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}^*, m)$
If $h = \perp$, return $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(h, m)\}$
return (h, r)
return 1, if $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{true} \wedge$
$\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m'^*, r'^*, h^*) = \mathsf{true} \wedge$
$(h^*, m^*) \notin \mathcal{Q} \ \wedge \ (h^*, \cdot) \in \mathcal{Q}$
return 0

Figure 6.6: CHET Private Collision-Resistance

Private Collision-Resistance. Private collision resistance requires that even the holder of the secret key $\mathsf{sk}_{\mathsf{CHET}}$ cannot find collisions as long as sktd is unknown. This is formalized by a honest hashing oracle which does not return sktd. Hence, $\mathcal{A}$'s goal is to return an actual collision on a non-adversarially generated hash h, for which it does not know sktd to avoid trivial wins of the adversary.

Experiment $\mathsf{Uniqueness}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$
$(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}_{\mathsf{CHET}})$
return 1, if $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathsf{true} \wedge r^* \neq r'^*$
return 0

Figure 6.7: CHET Uniqueness

Definition 6.14 (Private Collision-Resistance)**.** *A chameleon-hash with ephemeral trapdoor* CHET *is privately collision-resistant, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{PrivateCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.6.

Uniqueness. Uniqueness requires that it is hard to come up with two different randomness values for the same message m^* and hash value h^*, where pk^* is adversarially chosen, i.e., only the public parameters $\mathsf{pp}_{\mathsf{CHET}}$ are fixed.

Definition 6.15 (Uniqueness)**.** *A chameleon-hash with ephemeral trapdoor* CHET *is unique, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{Uniqueness}^{\mathsf{CHET}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 6.7.

Definition 6.16 (Secure Chameleon-Hashes with Ephemeral Trapdoors)**.** *A chameleon-hash with ephemeral trapdoor* CHET *is said to be secure, if it is correct, indistinguishable, publicly collision-resistant and privately collision-resistant.*

Note, is it not required that a secure CHET is unique, as it depends on the use-case whether this strong security notion is required.

6.3 Constructions

Regarding constructions of CHET schemes, it is natural to ask the question whether CHETs can be built from existing primitives in a black-box way. Interestingly, it is now shown how to elegantly "bootstrap" a CHET scheme in a black-box fashion from *any* existing secure (and unique) chameleon-hash, e.g., the one introduced before. If one does not require uniqueness, one can, e.g., resort to the recent scheme given by Ateniese et al. [AMVA17].

Then, direct constructions are given, two based on the DL assumption, and one based on an RSA-like assumption. While the DL-based constructions are not unique, the construction from RSA-like assumptions even achieves uniqueness. Note, however, that this strong security notion is not required in all use-cases. For example, in the application scenario given in Section 6.5, the CHETs do not need to be unique.

6.3.1 Black-Box Construction: Bootstrapping

Now a black-box construction from any existing chameleon-hash is presented. Namely, it is shown how one can achieve the desired goals by combining two instances of a secure chameleon-hash CH.

Construction 6.17 (Bootstrapped Construction)**.** *Obvious checks are omitted for brevity. Let* CHET *be defined as:*

$\mathsf{PPGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CHET}}$ *generate the public parameters in the following way:*

1. *Return* $\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$.

$\mathsf{KeyGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{CHET}}$ *generates the key pair in the following way:*

1. *Return* $(\mathsf{sk}^1_{\mathsf{CH}}, \mathsf{pk}^1_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CHET}})$.

$\mathsf{Hash}_{\mathsf{CHET}}$**.** *To hash a message* m*, w.r.t. public key* $\mathsf{pk}^1_{\mathsf{CH}}$ *do:*

1. *Let* $(\mathsf{sk}^2_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CHET}})$.
2. *Let* $(h_1, r_1) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, m)$.
3. *Let* $(h_2, r_2) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, m)$.
4. *Return* $((h_1, h_2, \mathsf{pk}^2_{\mathsf{CH}}), (r_1, r_2), \mathsf{sk}^2_{\mathsf{CH}})$.

$\mathsf{CheckHash}_{\mathsf{CHET}}$**.** *To check whether a given hash* $h = (h_1, h_2, \mathsf{pk}^2_{\mathsf{CH}})$ *is valid on input* $\mathsf{pk}_{\mathsf{CH}} = \mathsf{pk}^1_{\mathsf{CH}}$*,* m*,* $r = (r_1, r_2)$*, do:*

1. *Let* $b_1 \leftarrow \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, m, r_1, h_1)$.
2. *Let* $b_2 \leftarrow \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, m, r_2, h_2)$.
3. *If* $b_1 = \mathsf{false} \ \vee \ b_2 = \mathsf{false}$*, return* false.
4. *Return* true.

$\mathsf{Adapt}_{\mathsf{CHET}}$**.** *To find a collision w.r.t.* m*,* m'*, randomness* $r = (r_1, r_2)$*, hash* $h = (h_1, h_2, \mathsf{pk}^2_{\mathsf{CH}})$*,* $\mathsf{sktd} = \mathsf{sk}^2_{\mathsf{CH}}$*, and* $\mathsf{sk}_{\mathsf{CH}} = \mathsf{sk}^1_{\mathsf{CH}}$ *do:*

1. *If* $\mathsf{false} = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h)$*, return* $\perp$.
2. *Compute* $r'_1 \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}^1_{\mathsf{CH}}, m, m', r_1, h_1)$.
3. *Compute* $r'_2 \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}^2_{\mathsf{CH}}, m, m', r_2, h_2)$.
4. *Return* (r'_1, r'_2).

This construction is easy to understand, and only uses standard primitives. However, it remains to prove the following theorem.

Theorem 6.18. *If* CH *is secure and unique, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 6.17 is secure, and unique.*

Proof. Correctness follows from inspection; the remaining properties are proven below.

Indistinguishability. This follows by a simple argument. In particular, consider the following sequence of games.

Game 0: The original indistinguishability game, where $b = 0$.

Game 1: As Game 0, but instead of calculating the hash h_1 as in the game, directly hash. Due to the indistinguishability of the chameleon hashes, this hop only changes the view of the adversary negligibly due to the the indistinguishability of the chameleon hashes. More formally, assume that the adversary can distinguish this hop. One can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of the chameleon hashes. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as its own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}^1_{\mathsf{CH}}$, and proceeds as in the prior hop, with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate h_1. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

Game 2: As Game 1, but instead of calculating the hash h_2 as in the game, directly hash. Due to the indistinguishability of the chameleon hashes, this hop only changes the view of the adversary negligibly due to the the indistinguishability of the chameleon hashes. More formally, assume that the adversary can distinguish this hop. One can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of the chameleon hashes. In particular, the reduction works as follows. $\mathcal{B}$ receives pk'_c as its own challenge, $\mathcal{B}$ embeds pk'_c as $\mathsf{pk}^2_{\mathsf{CH}}$, and proceeds as in the prior hop, with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate h_2. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Clearly, this now case $b = 1$.

As each hop changes the view only negligibly, indistinguishability is proven. As each hop only changes the view of the adversary negligibly, this proves that the construction is indistinguishable.

Public Collision-Resistance. Let $\mathcal{A}$ be an adversary which breaks the public-collision resistance of the construction. One can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the collision-resistance of the underlying chameleon hash. This proven by a sequence of games:

Game 0: The original public collision-resistance game.

Game 1: As Game 0, but abort if the adversary $\mathcal{A}$ outputs a forgery $(m^*, r^*, m'^*, r'^*, h^*)$. Refer to E_1 as the abort event. A distinguisher $\mathcal{A}$ for this hop can be turned into a forger $\mathcal{B}$ against the collision-resistance of the underlying chameleon-hash. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}^1_{\mathsf{CH}}$ as the challenge public key. It uses this key to initialize $\mathcal{A}$. As the only oracle $\mathcal{B}$ has to simulate is the $\mathsf{Adapt}_{\mathsf{CHET}}$-oracle, it proceeds as follows. On input $m^*, m'^*, r^*, \mathsf{sktd}^*, h^*$, $\mathcal{B}$ first checks, if the hash verifies. If not, it returns $\perp$. Otherwise, $\mathcal{B}$ computes $r'_2 \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}^{2^*}_{\mathsf{CH}}, m^*, m'^*, r^*)$. Adversary $\mathcal{B}$ then queries its own adaption oracle to receive r'_1, and gives (r'_1, r'_2) to $\mathcal{A}$. At some point, $\mathcal{A}$ returns $(m^*, r^*, m'^*, r'^*, h^*)$. Via assumption, it is known that m^*, r^* w.r.t. m'^*, r'^* is "fresh", i.e., has never been returned by the adaption oracle. Thus, $\mathcal{B}$ can return $(m^*, r^*_1, m'^*, r'^*_1, h^*_1)$ as its own forgery attempt.

As each game hop only changes the view of the adversary negligibly, and the adversary has no way to forge a collision in Game 1, this proves that the construction is unique.

Private Collision-Resistance. The following sequence of games is used to prove the private collision-resistance of the construction.

Game 0: The original private collision-resistance game.

Game 1: As GAME 0, but the challenger aborts, if the adversary $\mathcal{A}$ outputs a valid forgery $(m^*, r^*, m'^*, r'^*, h^*)$. Use E_1 to refer to the abort event. This can be reduced the security to the collision-resistance of the underlying chameleon-hash. Moreover, let q_h be the number of queries to the hashing oracle. One can now construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the collision-resistance of the underlying chameleon hash. First, $\mathcal{B}$ receives $\mathsf{pk}^2_{\mathsf{CH}}$ from its own challenger, and pk^* from $\mathcal{A}$. Then proceed as follows. Draw a random index $i \xleftarrow{\$} [1, q_h]$. For each query $j \neq i$, let $(\mathsf{pk}^{2,i}_{\mathsf{CH}}, \mathsf{sk}^{2,i}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(1^\lambda)$. On input m, compute $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^{2,i}_{\mathsf{CH}}, m)$. Otherwise, i.e., $i = j$, on input m, compute $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, m)$. Next, let $\mathsf{pk}^{2,j}_{\mathsf{CH}} \leftarrow \mathsf{pk}^2_{\mathsf{CH}}$. In both cases, $\mathcal{B}$ gives $((r', r), (h, \mathsf{pk}^{2,i}_{\mathsf{CH}}))$ to $\mathcal{A}$. At some point, $\mathcal{A}$ returns $(m^*, r^*, m'^*, r'^*, h^*)$. One now knows that $h^*_2 = h'^*_2$ (i.e, the hash must have been returned by the oracle by definition) and m^* must be fresh by assumption, $\mathcal{B}$ can return $(m^*, r^*_2, m'^*, r'^*_2, h^*_2)$ as its own forgery attempt, if the hash returned is the one the challenge was embedded in. Thus, the probability that $\mathcal{B}$ wins is the same as $\mathcal{A}$, divided by q_h, as $\mathcal{B}$ has to guess on which query the adversary $\mathcal{A}$ finds the collision.

As each game hop only changes the view of the adversary negligibly and the adversary has no other way to forge a collision, this proves that the construction is privately collision-resistant.

Uniqueness. Let $\mathcal{A}$ be an adversary which breaks the uniqueness of the construction. One can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the uniqueness of the underlying chameleon-hash. In particular, consider the following sequence of games:

Game 0: The original uniqueness game.

Game 1: As GAME 0, but abort if the adversary outputs a randomness $r^* \neq r'^*$, a public key pk^*, along with a message m, and some hash h^* such that $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathsf{true}$. Let us use E_1 to refer to the abort event. This case can be reduced to the case of the uniqueness of the underlying chameleon-hash. $\mathcal{B}$ proceeds as follows. It initializes $\mathcal{A}$ with 1^λ. At some point, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$. By construction, one knows that r^* is of the form (r^*_1, r^*_2), and that h^* of the form $(h^*_1, h^*_2, \mathsf{pk}'^*)$, and r'^* is of the form (r'^*_1, r'^*_2), respectively. Moreover, by assumption, it is known that $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*_1, h^*_1) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r'^*_1, h^*_1) = \mathsf{true}$, but also that $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}'^*, m^*, r^*_2, h^*_2) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}'^*, m^*, r'^*_2, h^*_2) = \mathsf{true}$. However, one also knows that $r'^*_1 \neq r^*_1$ or $r'^*_2 \neq r^*_2$. Thus, $\mathcal{B}$ can return $(\mathsf{pk}^*, m^*, r^*_1, r'^*_1)$, if $r^*_1 \neq r'^*_1$, or $(\mathsf{pk}'^*, m^*, r^*_2, r'^*_2)$, if $r'^*_2 \neq r^*_2$.

As each game hop only changes the view of the adversary negligibly, this proves that the construction is unique. □

The question is now, if one can also directly construct CHET, which is answered to the affirmative subsequently.

6.3.2 A First Direct Construction

Now, a direct construction in groups where the DLP is hard using some ideas related to Pedersen commitments [Ped91] is presented. In a nutshell, the long-term secret is the discrete logarithm x between two elements g and h (i.e., $g^x = h$) of the long-term public key, while the ephemeral trapdoor is the randomness of the "commitment". To prohibit that a seen collision allows to extract the long-term secret key x, both trapdoors are hidden in a NIZKPoK. To make the "commitment" equivocable, it is then again randomized. To avoid that the holder of $\mathsf{sk}_{\mathsf{CH}}$ needs to store state, the randomness is encrypted to a public key of a IND-CCA2 secure encryption scheme contained in $\mathsf{pk}_{\mathsf{CH}}$. Security then directly follows from the DL assumption, IND-CCA2, the collision-resistance of the used hash function, and the extractability property of the NIZKPoK system. For brevity it is assumed that the NP-languages involved in the NIZKPoK are implicitly defined by the scheme. Note, this construction is not unique.

Construction 6.19 (CHET in Known-Order Groups)**.** *Let the set $\{\mathcal{H}^k_{\mathbb{Z}_q^*}\}_{k\in\mathcal{K}}$ denote a family of collision-resistant hash functions $\mathcal{H}^k_{\mathbb{Z}_q^*} : \{0,1\}^* \to \mathbb{Z}_q^*$ indexed by a key $k \in \mathcal{K}$ and let* CHET *be as follows:*

$\mathsf{PPGen}_{\mathsf{CHET}}$**.** *The algorithm $\mathsf{PPGen}_{\mathsf{CHET}}$ generates the public parameters in the following way:*

1. *Let $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$.*
2. *Let $k \xleftarrow{\$} \mathcal{K}$ for the hash function.*
3. *Let $\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda, L)$.*[2]
4. *Return $((\mathbb{G}, g, q), k, \mathsf{crs}_{\mathsf{NIZKPoK}})$.*

$\mathsf{KeyGen}_{\mathsf{CHET}}$**.** *The algorithm $\mathsf{KeyGen}_{\mathsf{CHET}}$ generates the key pair in the following way:*

1. *Draw random $x \xleftarrow{\$} \mathbb{Z}_q^*$. Set $h \leftarrow g^x$.*
2. *Generate $\pi_{\mathsf{pk}} \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(x) : h = g^x\}$.*
3. *Let $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.*
4. *Return $((x, \mathsf{sk}_{\mathsf{ENC}}), (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}}))$.*

$\mathsf{Hash}_{\mathsf{CHET}}$**.** *To hash m w.r.t. $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ do:*

1. *Return $\perp$, if $h \notin \mathbb{G}^\times$.*
2. *If π_{pk} is not valid, return $\perp$.*
3. *Draw random $r \xleftarrow{\$} \mathbb{Z}_q^*$.*
4. *Draw random $\mathsf{sktd} \xleftarrow{\$} \mathbb{Z}_q^*$.*
5. *Let $h' \leftarrow g^{\mathsf{sktd}}$.*
6. *Generate $\pi_t \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\mathsf{sktd}) : h' = g^{\mathsf{sktd}}\}$.*
7. *Encrypt r, i.e., let $C \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, r)$.*
8. *Let $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.*
9. *Let $p \leftarrow h^r$.*

[2] Actually one $\mathsf{crs}_{\mathsf{NIZKPoK}}$ is needed per language, but this is not made explicit here.

10. *Generate* $\pi_p \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r) : p = h^r\}$.
11. *Let* $b \leftarrow ph'^a$.
12. *Return* $((b, h', \pi_t), (p, C, \pi_p), \mathsf{sktd})$.

$\mathsf{CheckHash}_{\mathsf{CHET}}$. *To check whether a given hash* (b, h', π_t) *is valid on input* $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$, $m, r = (p, C, \pi_p)$, *do:*

1. *Return* false, *if* $p \notin \mathbb{G}^\times \ \vee \ h' \notin \mathbb{G}^\times$.
2. *If either* π_p, π_t, *or* π_{pk} *are not valid, return* $\perp$.
3. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
4. *Return* true, *if* $b = ph'^a$.
5. *Return* false.

$\mathsf{Adapt}_{\mathsf{CHET}}$. *To find a collision w.r.t.* m, m', (b, h', π_t), *randomness* (p, C, π_p), *and trapdoor information* sktd, *and* $\mathsf{sk}_{\mathsf{CH}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ *do:*

1. *If* $\mathsf{false} = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, m, (p, C, \pi_p), (b, h', \pi_t))$, *return* $\perp$.
2. *Decrypt* C, *i.e.,* $r \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, C)$. *If* $r = \perp$, *return* $\perp$.
3. *If* $h' \neq g^{\mathsf{sktd}}$, *return* $\perp$.
4. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
5. *Let* $a' \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m')$.
6. *If* $p \neq g^{xr}$, *return* $\perp$.
7. *If* $a = a'$, *return* (p, C, π_p).
8. *Let* $r' \leftarrow \frac{rx + a \cdot \mathsf{sktd} - a' \cdot \mathsf{sktd}}{x}$.
9. *Let* $p' \leftarrow h^{r'}$.
10. *Encrypt* r', *i.e., let* $C' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, r')$.
11. *Generate* $\pi'_p \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r') : p' = h^{r'}\}$.
12. *Return* (p', C', π'_p).

Some of the checks can already be done in advance, e.g., at a PKI, which only generates certificates, if the restrictions on each public key are fulfilled.

Theorem 6.20. *If the DL assumption in* $\mathbb{G}$ *holds,* $\mathcal{H}^k_{\mathbb{Z}^*_{|G|}}$ *is collision-resistant,* ENC *is IND-CCA2 secure, and* NIZKPoK *is secure, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 6.19 is secure.*

For the following proof it is assumed that the reduction sets up the NIZKPoK-parameters, but also the groups used in the protocol and the hashing-key k. This is not made explicit.

Proof. As before, it is only required to prove that the construction is indistinguishable, publicly collision-resistant, and privately collision-resistant. Again, each property is proven on its own.

Indistinguishability. Indistinguishability is trivial: as adversary never sees both the hash and the adapted hash at the same time the distributions are equivalent.

Public Collision-Resistance. Public collision-resistance is proven by a sequence of games.

Game 0: The original public collision-resistance game.

Game 1: As GAME 0, but obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau) \xleftarrow{\$} \mathsf{SIM}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \cdot)$. A distinguisher between GAME 0 and GAME 1 is a zero-knowledge distinguisher. A fully-fledged reduction is trivial and therefore omitted.

Game 2: As GAME 1, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \xi) \xleftarrow{\$} E_1(1^\lambda)$, and additionally store ξ. Under simulation sound extractability, this change is conceptual. Note, the values from the proofs are not extracted yet.

Game 3: As GAME 2, but additionally change the simulation of the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous query, set $\mathsf{AD} = \top$, and $\mathsf{AD} = \bot$ otherwise
2. If only C corresponds to a previous query, return $\bot$.
3. If $\mathsf{false} = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, (p, C, \pi_p), (b, h', \pi_t))$, return $\bot$.

...

This change is purely conceptual and thus does not change the view of the adversary at all. Observe that C is unconditionally binding, and, thus, modifying p implies that the check $p = g^{xr}$ which is performed within $\mathsf{Adapt}_{\mathsf{CHET}}$ fails and the oracle would abort anyway in this case.

Game 4: Behave exactly as in GAME 3, but further change the simulation of the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous $\mathsf{Adapt}_{\mathsf{CHET}}$ query, set $\mathsf{AD} = \top$ and $\mathsf{AD} = \bot$ otherwise. If only C corresponds to a previous query, return $\bot$.

...

3. If $\mathsf{AD} = \bot$, decrypt C, i.e., $r \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, C)$. If the resulting decryption is invalid, i.e., $r = \bot$, return $\bot$.

...

7. If $\mathsf{AD} = \bot$, check if $p \neq g^{xr}$, and return $\bot$ if so.

...

This change is conceptual (the checks are only omitted if one knows that they would not yield to an abort).

Game 5: Behave exactly as in GAME 4, but further change the simulation of the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous $\mathsf{Adapt}_{\mathsf{CHET}}$ query, set $\mathsf{AD} = \top$ and $\mathsf{AD} = \bot$ otherwise. If only C corresponds to a previous query, return $\bot$.

...

8. If $a = a'$, return (p, C, π_p).
9. $\boxed{\texttt{NOP}}$ //NOP means NO Operation
10. Let $\boxed{p' \leftarrow \frac{b}{g^{a' \mathsf{sktd}}}}$.
11. $\boxed{C' \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, 0)}$.
12. Generate $\pi'_p \xleftarrow{\$} \mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, (p', h))$.

...

A distinguisher between GAME 3 and GAME 4 is an IND-CCA2 distinguisher for ENC, using a standard hybrid argument.

Game 5: As GAME 4, but further modify $\mathsf{Adapt}_{\mathsf{CHET}}$ so that it runs on an $\mathsf{sk}_{\mathsf{ch}}$ where x is replaced by g^x:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (\boxed{g^x}, \mathsf{sk}_{\mathsf{ENC}})$ do:

...

7. If $\mathsf{AD} = \bot$, check if $p \neq (\boxed{g^x})^r$, and return $\bot$ if so.

...

This change is conceptual.

Game 6: As GAME 5, but for every query to $\mathsf{Adapt}_{\mathsf{CHET}}$, store (p, C, π_p), if π_p was not previously simulated within $\mathsf{Adapt}_{\mathsf{CHET}}$ in $\mathtt{R}[(b, h', \pi_t)] \leftarrow (p, C, \pi_p)$. Now, for every forgery either both r^* or r'^* are fresh, or one of them contains a proof π_p (resp. π'_p) which was previously simulated in the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle. If one of them contains such a proof, replace the respective randomness tuple (p, C, π_p) by $\mathtt{R}[h^*]$. This change is conceptual. Observe that the fact that a proof stems from a tuple returned by $\mathsf{Adapt}_{\mathsf{CHET}}$ implies that a query with a tuple (p, C, π_p) where π_p was not simulated must once have happened. Further, the modified forgery is still a valid public collision freeness forgery.

Game 7: As GAME 6, but for the modified forgery extract both r and r' from π_p and π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$. If the extraction fails, abort. Both games proceed identically, unless the abort event happens. Due to the extractability property of the proof system, this only happens with negligible probability. For simplicity, both extractions are collapsed in a single game change. It is easy to unroll them into two separate game changes, i.e., changing them in a hybrid game.

Game 8: As GAME 7, but for π_t contained in h^* extract sktd and abort if the extraction fails. Both games proceed identically, unless the abort event happens, which, due to the extractability property of the proof-system is negligible.

Game 9: As GAME 8, but obtain a DL-challenge $(\mathbb{G}, g, q, g^x)$, perform the setup with respect to $(\mathbb{G}, g, q)$ and embed g^x into $\mathsf{sk}_{\mathsf{ch}}$. This change is conceptual.

In GAME 9, for every forgery (modified according to GAME 6) it holds that $h^* = (b, h', \pi_t)$ contains $b = g^{rx+a\mathsf{sktd}} = g^{r'x+a'\mathsf{sktd}}$. Thus, it holds that $rx + a\mathsf{sktd} = r'x + a'\mathsf{sktd}$. Hence, x can easily be calculated, which is the solution to the DL-challenge. This extraction is only possible, if $a \neq a'$. However, if this is not the case there is a collision in the hash-function, which can easily be extracted. As the view between all intermediate games only changes negligibly, the proof is concluded.

Private Collision-Resistance. Below private collision resistance is proven using a sequence of games.

Game 0: The original private collision-resistance game.

Game 1: As GAME 0, but obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau) \xleftarrow{\$} \mathsf{SIM}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \cdot)$. A distinguisher between GAME 0 and GAME 1 is a zero-knowledge distinguisher.

Game 2: As GAME 1, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \xi) \xleftarrow{\$} E_1(1^\lambda)$, and additionally store ξ. Under simulation sound extractability, this change is conceptual.

Game 3: As GAME 2, but modify the $\mathsf{Hash}_{\mathsf{CHET}}$ oracle so that it no longer draws sktd uniformly at random, but directly draws h' uniformly at random from $\mathbb{G}^\times$.

$\mathsf{Hash}_{\mathsf{CHET}}$. To hash m w.r.t. $\mathsf{pk}_{\mathsf{CHET}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ do:

...
5. Let $\boxed{h' \xleftarrow{\$} \mathbb{G}^\times}$.
...

Game 4: As GAME 3, but for every π_p returned by $\mathsf{Hash}_{\mathsf{CHET}}$ record the value r so that $p = h^r$ in $\mathtt{R}[p] \leftarrow r$. This change is conceptual.

Game 5: As GAME 4, but for pk^* output by the adversary extract x so that $g^x = h$. If the extraction fails, abort. Both games proceed identically, unless there is the abort event, which only happens with negligible probability due to the extractability property.

Game 6: As GAME 5, but obtain a DL instance $(\mathbb{G}, g, q, g^t)$, perform the setup with respect to $(\mathbb{G}, g, q)$ and further modify $\mathsf{Hash}_{\mathsf{CHET}}$ as follows:

$\mathsf{Hash}_{\mathsf{CHET}}$. To hash m w.r.t. $\mathsf{pk}_{\mathsf{CHET}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ do:

...
5. Let $\boxed{s \xleftarrow{\$} \mathbb{Z}_q^*,\ h' \leftarrow (g^t)^s}$.
...

Furthermore, record $\mathtt{S}[h'] \leftarrow s$. This change is conceptual.

Game 7: As GAME 6, but if π_p or π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$ do not correspond to a $\mathsf{Hash}_{\mathsf{CHET}}$ answer, obtain r and r' using the extractor and set $\mathtt{R}[p] \leftarrow r$ or $\mathtt{R}[p'] \leftarrow r'$. If the extraction fails, abort. Both games proceed identical unless the abort event happens.[3]

Now, if the adversary $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ such that $h^* = g^{xr}h'^a = g^{xr'}h'^{a'}$ in GAME 7, $\mathcal{B}$ proceeds as follows. By definition, it holds that $g^{rx+ats} = g^{r'x+a'ts}$ (Note, $g^{ts} = h'$ in both cases, which, by definition, needs to be returned by the $\mathsf{Hash}_{\mathsf{CHET}}$ oracle, and thus $s = \mathtt{S}[h']$ is known). It follows that $rx + ats = r'x + a'ts$ holds. As all variables but t are now known (the values for r and r' can be obtained from $\mathtt{R}$), it also holds that t can be calculated and returned as DL solution unless $a = a'$, which would however imply a collision for the hash function. Thus, as the views of the adversary changes only changes negligibly, security of the construction is finally proven. □

6.3.3 A Direct Construction From RSA-Like Assumptions

Now, an extension to the RSA-based chameleon-hash given in Section 6.1.3 (which is itself based on the construction given by Brzuska et al. [BFF+09], see Appendix I for their construction) using the technique used for accumulators by Pöhls et al. [PPS+13]. In the construction, the trapdoor is an additional RSA-modulus N'. Only if the factorization of $N' = p'q'$, contained in sktd, and $N = pq$, which is the secret key $\mathsf{sk}_{\mathsf{CHET}}$, is known, a collision can be produced. It is assumed that the bit-length of N and N' is the same, which is implicitly given by the security parameter λ. Note, the condition $e > N^3$ implies that $e > NN'$, and thus $\gcd(\varphi(NN'), e) = 1$, which makes the analysis simpler (cf. Lemma 6.8).

This is not explicitly checked in the algorithms.

Construction 6.21 (CHET from RSA-like Assumptions)**.** *Let $\mathcal{H}_N : \{0,1\}^* \to \mathbb{Z}_N^*$, $N \in \mathbb{N}$, denote a random oracle. Let* CHET *be defined as:*

$\mathsf{PPGen}_{\mathsf{CHET}}$. *The algorithm* $\mathsf{PPGen}_{\mathsf{CHET}}$ *generates the public parameters in the following way:*

1. *Call* RSAGen *with the restriction $e > N^3$, and e prime. Return e.*

$\mathsf{KeyGen}_{\mathsf{CHET}}$. *The algorithm* $\mathsf{KeyGen}_{\mathsf{CHET}}$ *generates the key pair in the following way:*

[3] For simplicity both extractions are collapsed in a single game change. It is easy to unroll them into two separate game changes.

1. *Generate two primes p and q using* $\mathsf{RSAGen}(1^\lambda)$. *Set* $\mathsf{sk}_{\mathsf{CH}} \leftarrow (p,q)$. *Let* $N \leftarrow pq$. *Set* $\mathsf{pk}_{\mathsf{CH}} \leftarrow N$.
2. *Return* $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$.

$\mathsf{CheckHash}_{\mathsf{CHET}}$. *To hash a message m w.r.t.* $\mathsf{pk}_{\mathsf{CH}} = N$ *do:*

1. *Generate two primes p' and q' using* $\mathsf{RSAGen}(1^\lambda)$. *Set* $\mathsf{sktd} \leftarrow (p',q')$, *and* $N' \leftarrow p'q'$.
2. *If* $\gcd(N,N') \neq 1$, *go to* 1.
3. *Draw* $r \leftarrow \mathbb{Z}^*_{NN'}$.
4. *Let* $g \leftarrow \mathcal{H}_{NN'}(m)$, *and* $h \leftarrow gr^e \bmod NN'$.
5. *Return* $((h,N'),r,\mathsf{sktd})$.

$\mathsf{CheckHash}_{\mathsf{CHET}}$. *To check whether a given hash h is valid on input* $\mathsf{pk}_{\mathsf{CH}} = N$, *$m$, and r, do:*

1. *Return* $\perp$, *if* $r \notin \mathbb{Z}^*_{NN'}$.
2. *Let* $g \leftarrow \mathcal{H}_{NN'}(m)$, *and* $h' \leftarrow gr^e \bmod NN'$.
3. *Return* true, *if* $h = (h',N')$.
4. *Return* false.

$\mathsf{Adapt}_{\mathsf{CHET}}$. *To find a collision w.r.t. m, m', randomness r, hash h, trapdoor information* sktd, *and* $\mathsf{sk}_{\mathsf{CH}}$ *do:*

1. *Check that* $N' = p'q'$, *where p' and q' is taken from* sktd. *If this is not the case, return* $\perp$.
2. *If* $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) = \mathsf{false}$, *return* $\perp$.
3. *Compute d s.t.* $de \equiv 1 \bmod \varphi(NN')$.
4. *Let* $g \leftarrow \mathcal{H}_{NN'}(m)$, *and* $h \leftarrow gr^e \bmod NN'$.
5. *Let* $g' \leftarrow \mathcal{H}_{NN'}(m')$ *and* $r' \leftarrow (h(g'^{-1}))^d \bmod NN'$.
6. *Return* r'.

Theorem 6.22. *If the one-more RSA-inversion assumption holds, then the above construction is secure in the random-oracle model.*

Proof. Again, one needs to prove that the construction is indistinguishable, unique, publicly collision-resistant, and privately collision-resistant.

Indistinguishability. It is easy to see that the above construction is indistinguishable; all values are chosen uniformly at random and RSA defines a permutation. See also Lemma 6.8.

Public Collision-Resistance. Now is proven, via a sequence of games, that the above construction is collision-resistant.

Game 0: This is the original public collision-resistance game.

Game 1: As Game 0, but instead of using the e from the system parameters (here, $e > N^3$), Embed the e received from a one-more RSA challenger as $\mathsf{pp}_{\mathsf{CHET}}$. Note, one can still determine d—and therefore honestly simulate all oracles—as the N chosen by the challenger at this point is not used, and one can thus freely choose it. This does not change the view of the adversary, as the received e is distributed identically to the real game.

Game 2: As Game 1, but abort, if the adversary was able to generate a collision $(m^*, r^*, m'^*, r'^*, h^*)$. Let this event be denoted E_2. Assume that event E_2 does happen with non-negligible probability. One can then build an adversary $\mathcal{B}$ which breaks the one-more RSA-inversion assumption. Without loss of generality, assume that the adversary makes all the random oracle queries before outputting the messages (otherwise, $\mathcal{B}$ does them). The adversary $\mathcal{B}$ proceeds as follows. In the first step, the challenge N_c is embedded in $\mathsf{pk}_{\mathsf{CH}}$ as N. Clearly, as the distributions are the same, this is only a conceptual change so far. In the second step, for each *new* random-oracle query m_i to $\mathcal{H}_{NN'}$, $\mathcal{B}$ asks its challenge oracle $\mathcal{C}$ to provide a challenge $c_i \in \mathbb{Z}_N^*$. However, it may happen that $c_i \notin \mathbb{Z}_{NN'}^*$ (note, that one has a family of random-oracles!). In this case, request a new c_i from the challenge oracle till the condition holds.[4] Draw $u_i \leftarrow \mathbb{Z}_{NN'}^*$ and record $(m_i, N', c_i, u_i, \perp)$. Embed $c_i u_i^e \bmod NN'$ as the random-oracle response for m_i. Note, this value is distributed perfectly uniformly in $\mathbb{Z}_{NN'}^*$. However, it remains open how to simulate the adaption oracle. Assume, for now, that m_0 is supposed to be adapted to m_1, while the second modulus is N'. If $m_0 = m_1$, or $n = N'$, proceed as in the algorithm. If there is no tuple (m_0, N', c_0, u_0, z_0), with $z_0 \neq \perp$, query the inversion oracle with c_0 to receive z_0. Update record $(m_0, N', c_0, u_0, \perp)$ to (m_0, N', c_0, u_0, z_0). If there is no tuple (m_1, N', c_1, u_1, z_1), with $z_1 \neq \perp$, query the inversion oracle with c_1 to receive z_1. Update record $(m_1, N', c_1, u_1, \perp)$ to (m_1, N', c_1, u_1, z_1). Calculate the collision $\bmod N$ as $rz_0(z_1)^{-1}$. The collision $\bmod N'$ can be calculated honestly, as the factors are known. Combine the result $\bmod NN'$ using the Chinese remainder theorem, and return it. Eventually, the adversary returns $(m^*, r^*, m'^*, r'^*, h^*)$. Then, one knows (by construction) that $\mathcal{H}_{NN'}(m^*)r^{*e} \equiv \mathcal{H}_{NN'}(m'^*)r'^{*e} \bmod NN'$. If there is no record for m^* *and* no record for m'^*, query the inversion for oracle for the root z^* for $\mathcal{H}_{NN'}(m^*)$, and update record $(m^*, N', c^*, r^*, \perp)$ to (m^*, N', c^*, u^*, z^*). Then, by definition of the security game, one knows that m'^* is fresh, and there exists a record $(m'^*, N', c'^*, u'^*, \perp)$. One can then extract $\mathcal{H}_{NN'}(m'^*)^d$ by calculating $\mathcal{H}_{NN'}(m'^*)^d \equiv r'^{*-1}z^*r^* \bmod NN'$, and extract the root of c'^* by multiplying it with $u'^{*-1} \bmod NN'$, resulting in z'^*. As therefore the adversary $\mathcal{A}$ has inverted more challenges than the inversion oracle was queried, $\mathcal{B}$ can return the list $\{(c_i, z_i)\}$ for each entry where $(m_i, N_i', c_i, r_i, z_i)$, $z_i \neq \perp$ exists, along with $(c'^*, z'^* \bmod n)$.

As now the adversary has no other way to win its game, public collision-resistance is proven, as each hop only changes the view of the adversary negligibly.

Private Collision-Resistance. Now, via a sequence of games, is proven that the above construction is collision-resistant.

Game 0: This is the original private collision-resistance game.

[4] Lemma 6.9 directly provides a polynomial bound on the expected number of needed samples. To have a strictly polynomial-time $\mathcal{B}$, abort if one needs $\mathsf{p}(\lambda)$ times more samples than expected, for some fixed polynomial p. Clearly, this only happens with at most negligible probability, and therefore does not significantly change the winning probability in the following.

Game 1: As GAME 0, but instead of using the e from the system parameters (here, $e > N^3$), Embed the e received from a RSA challenger as $\mathsf{pp}_{\mathsf{CHET}}$. As before, one can still determine d—and therefore honestly simulate all oracles—as the N' from the challenger is not embedded at this point, and one can thus freely choose it. This does not change the view of the adversary, as the received e is distributed identically to the real game.

Game 2: As GAME 1, but now abort, if there are random-oracle collisions in any random oracle. Let this event be E_2. Event E_2 cannot happen with non-negligible probability due to the birthday-bound.

Game 3: Now abort, if the adversary was able to output a tuple $(m^*, r^*, m'^*, r'^*, h^*)$ which breaks the private collision-resistance of the construction. Let this event be E_3. First, note that without loss of generality one only needs to consider such adversaries $\mathcal{A}$ that only make a single call to the $\mathsf{CheckHash}'_{\mathsf{CHET}}$ oracle, as it can simulate all other calls (except for the h^*) internally.[5]

Assume now that the abort event happens with non-negligible probability. One can then construct an adversary $\mathcal{B}$ which breaks the one-more RSA assumption with non-negligible probability. $\mathcal{B}$ simulates the $\mathsf{CheckHash}'_{\mathsf{CHET}}$ oracle by embedding the modulus it received from its own RSA challenger as N'.[6] For computing $\mathcal{H}_{NN'}(m)$, it asks its own challenger for challenges in $\mathbb{Z}^*_{N'}$ until it receives a challenge c that also lies in $\mathbb{Z}^*_{NN'}$ (as before, this happens after a polynomial number of steps by Lemma 6.9). It then chooses a random $u \leftarrow \mathbb{Z}^*_{NN'}$, sets $\mathcal{H}_{NN'}(m) = cu^e \bmod NN'$, and computes its response as in the original algorithm, i.e., it outputs $((\mathcal{H}_{NN'}(m)r^e \bmod NN', N'), r)$ for a random r. All other queries to the $\mathcal{H}_{NN'}(m_j)$ oracle are replied by v_j^e for a fresh $v_j \leftarrow \mathbb{Z}^*_{NN'}$, and the pairs (m_j, v_j) are stored internally. As before, assume that $\mathcal{A}$ did all the random oracle queries before it outputs its messages (otherwise, $\mathcal{B}$ makes the necessary queries). Eventually, $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ with $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m'^*, r'^*, h^*) = \mathsf{true}$ with h^* as returned by $\mathcal{B}$ and $m'^* \neq m^*$. $\mathcal{B}$ then looks up v^* such that $\mathcal{H}_{NN'}(m^*) = v^{*e}$. By assumption it then holds that $\mathcal{H}_{NN'}(m^*)r^{*e} = \mathcal{H}_{NN'}(m'^*)r'^{*e}$, i.e., that $v^{*e}r^{*e} = cu^e r^e \bmod NN'$, or equivalently that $c = (v^*r^*u^{-1}r^{-1})^e \bmod NN'$. $\mathcal{B}$ now outputs $(c, x = v^*r^*u^{-1}r^{-1} \bmod N')$ and returns it to its one-more RSA challenger. It is easy to see that $c = x^e \bmod N'$ holds: $c = (v^*r^*u^{-1}r^{-1})^e \bmod NN'$ means that $c = (v^*r^*u^{-1}r^{-1})^e + kNN'$ for some integer k. Reducing by N' yields $c = (v^*r^*u^{-1}r^{-1})^e \bmod N'$. As $\mathcal{B}$ did not query the inversion oracle at all, it thus wins with a probability only polynomially smaller than E_3, contradicting the assumption that E_3 is not negligible. The private collision-resistance follows. Note that the private collision resistance property actually already holds under the standard RSA assumption (not the one-more RSA assumption), as $\mathcal{B}$ never queries the inversion oracle. Formally, in the proof, it would abort when receiving a $c \notin \mathbb{Z}^*_{NN'}$, which—by Lemma 6.9—imposes a polynomial loss.

Uniqueness. Now is proven that the above construction is unique using a sequence of games:

[5] Formally, in the following $\mathcal{B}$ honestly simulates all calls to $\mathsf{CheckHash}'_{\mathsf{CHET}}$, except for a random query where it embeds the challenge; this causes a loss of $1/q_h$ in the success probability, where q_h is the number of oracle queries made by $\mathcal{A}$.

[6] Note that the adversary already has access to the random oracles in its first phase, i.e., before outputting pk^*. In the following, assume that it never queried the oracle for $\mathcal{H}_{NN'}(\cdot)$ during this phase. Given the super-polynomial number of possible values for N', this only introduces a negligible loss in the following.

Game 0: The original uniqueness game.

Game 1: As GAME 0, but the challenger aborts, if the adversary finds randomness $r^* \neq r'^*$, a public key $\mathsf{pk}^* = N$, a message m^*, and a hash h^* such that $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathsf{true}$. Let this abort event be denoted E_1. This cannot happen, as RSA (with the given restrictions on e and r) defines a permutation, and random-oracles behave as functions, regardless of the choice of N. See also Lemma 6.8. This proves that the construction is unique. □

6.3.4 A Construction in Gap-Groups

Next, a a second direct construction in prime order groups equipped with a bilinear map is given. This construction is derived from the first construction. Essentially, the main idea is to use a DDH oracle to check the correctness of the commitment. Subsequently, the construction is presented, where it is assumed that the NP-languages involved in the proofs of knowledge are implicitly defined by the scheme.

Construction 6.23 (CHET in Gap-Groups)**.** *Let $\{\mathcal{H}^k_{\mathbb{Z}_q^*}\}_{k\in\mathcal{K}}$ denote a family of collision-resistant hash functions $\mathcal{H}^k_{\mathbb{Z}_q^*} : \{0,1\}^* \to \mathbb{Z}_q^*$ indexed by a key $k \in \mathcal{K}$ and let* CHET *be defined as:*

$\mathsf{PPGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{PPGen}_{\mathsf{CHET}}$ *generates the public parameters in the following way:*

1. *Let* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda)$.
2. *Let* $k \xleftarrow{\$} \mathcal{K}$ *for the hash function.*
3. *Let* $\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda)$.[7]
4. *Return* $((\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q), k, \mathsf{crs}_{\mathsf{NIZKPoK}})$

$\mathsf{KeyGen}_{\mathsf{CHET}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{CHET}}$ *generates the key pair in the following way:*

1. *Draw random* $x \xleftarrow{\$} \mathbb{Z}_q^*$. *Set* $h \leftarrow g_2^x$.
2. *Generate* $\pi_{\mathsf{pk}} \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(x) : h = g_2^x\}$.
3. *Let* $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.
4. *Return* $((x, \mathsf{sk}_{\mathsf{ENC}}), (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}}))$.

$\mathsf{Hash}_{\mathsf{CHET}}$**.** *To hash m w.r.t.* $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ *do:*

1. *If* π_{pk} *is not valid, return* $\perp$.
2. *Draw random* $r \xleftarrow{\$} \mathbb{Z}_q^*$.
3. *Draw random* $\mathsf{sktd} \xleftarrow{\$} \mathbb{Z}_q^*$.
4. *Let* $h' \leftarrow g_2^{\mathsf{sktd}}$.
5. *Generate* $\pi_t \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\mathsf{sktd}) : h' = g_2^{\mathsf{sktd}})\}$.

[7] Actually one $\mathsf{crs}_{\mathsf{NIZKPoK}}$ is needed per language, but this is not make explicit here.

6. *Encrypt r, i.e., let* $C \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, r)$.
7. *Let* $p \leftarrow e(g_1^r, h)$.
8. *Generate* $\pi_p \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(r) : p = e(g_1^r, h)\}$.
9. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
10. *Let* $b \leftarrow p \cdot e(g_1^a, h')$.
11. *Return* $((b, h', \pi_t), (p, C, \pi_p), \mathsf{sktd})$.

$\mathsf{CheckHash}_{\mathsf{CHET}}$. *To check whether a given hash* (b, h', π_t) *is valid on input* $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$, m, (p, C, π_p) *do:*

1. *Return* false, *if* $p \notin \mathbb{G}_T^\times \ \vee \ h' \notin \mathbb{G}_2^\times$.
2. *If either* π_p, π_t, *or* π_{pk} *are not valid, return* $\perp$.
3. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m, \tau)$.
4. *Return* true, *if* $b = p \cdot e(g_1^a, h')$.
5. *Return* false.

$\mathsf{Adapt}_{\mathsf{CHET}}$. *To find a collision w.r.t.* m, m', *randomness* (p, C, π_p), *and trapdoor information* sktd, *and* $\mathsf{sk}_{\mathsf{CH}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ *do:*

1. *If* $\mathsf{false} = \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, m, (p, C, \pi_p), (b, h', \pi_t))$, *return* $\perp$.
2. *Return* $\perp$, *if* $h' \neq g_2^{\mathsf{sktd}}$.
3. *Decrypt* C, *i.e.,* $r \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, C)$. *If* $r = \perp$, *return* $\perp$.
4. *If* $m = m'$, *return* (p, C, π_p).
5. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
6. *Let* $a' \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m')$.
7. *If* $p \neq e(g_1^r, g_2^x)$, *return* $\perp$.
8. *If* $a = a'$, *return* $r = (p, C, \pi_p)$.
9. *Let* $r' \leftarrow \frac{rx + a \cdot \mathsf{sktd} - a' \cdot \mathsf{sktd}}{x}$.
10. *Let* $p' \leftarrow e(g_1^{r'}, g_2^x)$.
11. *Encrypt* r', *i.e., let* $C' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, r')$.
12. *Generate* $\pi'_p \xleftarrow{\$} \mathsf{NIZKPoK}\{(r') : p' = e(g_1^{r'}, g_2^x)\}$.
13. *Return* (p', C', π'_p).

Most of the checks can already be done in advance, e.g., at a PKI, which only generates certificates, if the restrictions on each public key are fulfilled. Moreover, it is not required that the correctness of the ciphertext is proven.

Theorem 6.24. *If the DL assumption in* $\mathbb{G}_2$ *holds,* $\mathcal{H}^k_{\mathbb{Z}^*_{|\mathbb{G}_2|}}$ *is collision-resistant,* ENC *is IND-CCA2 secure, and* NIZKPoK *is secure, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 6.23 is secure.*

Proof. One needs to prove that the construction is indistinguishable, publicly collision-resistant, and privately collision-resistant.

Indistinguishability. Indistinguishability is trivial: as the adversary never sees both the hash and the adapted hash at the same time, the distributions are identical. The proof is therefore omitted.

Public Collision-Resistance. Public collision-resistance is proven by a sequence of games.

Game 0: The original public collision-resistance game.

Game 1: As Game 0, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau) \xleftarrow{\$} \mathsf{SIM}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \cdot)$. A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher.

Game 2: As Game 1, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \xi) \xleftarrow{\$} E_1(1^\lambda)$, and additionally store ξ. Under simulation sound extractability, this change is conceptual.

Game 3: As Game 2, but simulate the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous query, set $\mathsf{AD} = \top$, and $\mathsf{AD} = \bot$ otherwise
2. If only C corresponds to a previous query, return $\bot$.
3. If $\mathsf{false} = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, (p, C, \pi_p), (b, h', \pi_t))$, return $\bot$.

...

This change is conceptual. Observe that C is unconditionally binding, and, thus, modifying p implies that the check $p = e(g_1^x, g_2^r)$ which is performed within $\mathsf{Adapt}_{\mathsf{CHET}}$ fails, and the oracle would abort anyway).

Game 4: As Game 3, but further change the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CH}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous $\mathsf{Adapt}_{\mathsf{CHET}}$ query, set $\mathsf{AD} = \top$ and $\mathsf{AD} = \bot$ otherwise. If only C corresponds to a previous query, return $\bot$.

...

3. If $\mathsf{AD} = \bot$, decrypt C, i.e., $r \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, C)$. If $r = \bot$, return $\bot$.

...

7. If $\mathsf{AD} = \bot$, check if $p \neq e(g_1^r, g_2^x)$, and return $\bot$ if so.

...

This change is conceptual, as the checks are only omitted if they would not yield to an abort.

Game 4: As GAME 3, but further change the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle as follows:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CH}} = (x, \mathsf{sk}_{\mathsf{ENC}})$ do:

1. If $(p, C, \cdot)$ corresponds to a previous $\mathsf{Adapt}_{\mathsf{CHET}}$ query, set $\mathsf{AD} = \top$ and $\mathsf{AD} = \bot$ otherwise. If only C corresponds to a previous query, return $\bot$.

...

8. If $a = a'$, return (p, C, π_p).
9. $\boxed{\texttt{NOP}}$
10. Let $\boxed{p' \leftarrow \frac{b}{e(g^{a'}, h')}}$.
11. $\boxed{C' \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, 0)}$.
12. Generate $\pi'_p \leftarrow \mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, (p', g_2^x))$.

...

A distinguisher between GAME 3 and GAME 4 is an IND-CCA2 distinguisher for ENC, using a standard hybrid argument.

Game 5: As GAME 4, but further modify $\mathsf{Adapt}_{\mathsf{CHET}}$ so that it runs on an $\mathsf{sk}_{\mathsf{CHET}}$ where x is replaced by g^x:

$\mathsf{Adapt}_{\mathsf{CHET}}$. To find a collision w.r.t. m, m', (b, h', π_t), randomness (p, C, π_p), and trapdoor information sktd, and $\mathsf{sk}_{\mathsf{CHET}} = (\boxed{g_2^x}, \mathsf{sk}_{\mathsf{ENC}})$ do:

...

This change is conceptual.

Game 6: As GAME 5, but for every query to $\mathsf{Adapt}_{\mathsf{CHET}}$, store (p, C, π_p) if π_p was not previously simulated within $\mathsf{Adapt}_{\mathsf{CHET}}$ in $\mathtt{R}[(b, h', \pi_t)] \leftarrow (p, C, \pi_p)$. Now, for every forgery either both r^* or r'^* are fresh, or one of them contains a proof π_p (resp. π'_p) which was previously simulated in the $\mathsf{Adapt}_{\mathsf{CHET}}$ oracle. If one of them contains such a proof, replace the respective randomness tuple (p, C, π_p) by $\mathtt{R}[h^*]$. This change is conceptual. Observe that the fact that a proof stems from a tuple returned by $\mathsf{Adapt}_{\mathsf{CHET}}$ implies that a query with a tuple (p, C, π_p) where π_p was not simulated must once have happened. Further, the modified forgery is still a valid public collision freeness forgery.

Game 7: As GAME 6, but for the modified forgery extract both r and r' from π_p and π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$. If the extraction fails, abort. Both games proceed identically, unless the abort event happens. This event only happens with negligible probability due to the extractability property of the proof system. For simplicity both extractions were collapsed in a single game change. It is easy to unroll them into two separate game changes.

Game 8: As GAME 7, but for π_t contained in h^* extract sktd, and abort if the extraction fails. Both games proceed identically, unless the abort event happens.

Game 9: As GAME 8, but obtain a DL-challenge $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, g_2^x)$, perform the setup with respect to $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q)$ and embed g_2^x into $\mathsf{pk}_{\mathsf{CHET}}$. This change is conceptual.

In GAME 9, for every forgery (modified according to GAME 6) it holds that $h^* = (b, h', \pi_t)$ contains $b = g_T^{rx+a\mathsf{sktd}} = g_T^{r'x+a'\mathsf{sktd}}$. Thus, $rx + a\mathsf{sktd} = r'x + a'\mathsf{sktd}$ follows. Hence, x can easily be calculated, which is the solution to the DL-challenge. This extraction is only possible, if $a \neq a'$. However, if this is not the case, there is a collision in the hash-function. Taking the union bound yields that $\Pr[S_9] = \nu_{\mathsf{DL}}(\lambda) + \nu_{\mathsf{CR}}(\lambda)$; all intermediate game changes are negligible, which concludes the proof.

Private Collision-Resistance. Below private collision resistance is proven, using a sequence of games.

Game 0: The original private collision-resistance game.

Game 1: As GAME 0, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau) \leftarrow \mathsf{SIM}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathsf{SIM}_2(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \cdot)$. A distinguisher between GAME 0 and GAME 1 is a zero-knowledge distinguisher.

Game 2: As GAME 1, but upon setup obtain $(\mathsf{crs}_{\mathsf{NIZKPoK}}, \tau, \xi) \leftarrow E_1(1^\lambda)$, and additionally store ξ. Under simulation sound extractability, this change is conceptual, and thus does not change the distribution.

Game 3: As GAME 2, but modify the $\mathsf{Hash}_{\mathsf{CHET}}$ oracle so that it no longer draws sktd uniformly at random but directly draws h' uniformly at random from $\mathbb{G}_2^*$.

$\mathsf{Hash}_{\mathsf{CHET}}$. To hash m w.r.t. $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ do:

...
5. Let $\boxed{h' \xleftarrow{\$} \mathbb{G}_2^\times}$.
...

Game 4: As GAME 3, but for every π_p returned by $\mathsf{Hash}_{\mathsf{CHET}}$, also record the value r so that $p = e(g_1^r, h)$ in $\mathtt{R}[p] \leftarrow r$. This change is conceptual.

Game 5: As GAME 4, but for pk^* output by the adversary one can extract x so that $g_2^x = h$. If the extraction fails, abort. Both games proceed identically, unless the abort event happens. This can only happen with negligible probability due to the extractability property of the proof system.

Game 6: As GAME 5, but obtain a DL instance $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, g_2^t)$, perform the setup with respect to $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q)$ and further modify $\mathsf{Hash}_{\mathsf{CHET}}$ as follows:

$\mathsf{Hash}_{\mathsf{CHET}}$. To hash m w.r.t. $\mathsf{pk}_{\mathsf{CH}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{ENC}})$ do:

...
5. Let $\boxed{s \leftarrow \mathbb{Z}_q^*,\ h' \leftarrow (g_2^t)^s}$.
...

Furthermore, keep the record $\mathtt{S}[h'] \leftarrow s$. This change is conceptual.

Game 7: As Game 6, but if π_p or π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$ do not correspond to a $\mathsf{Hash}_{\mathsf{CHET}}$ answer obtain r and r' using the extractor and set $\mathtt{R}[p] \leftarrow r$ or $\mathtt{R}[p'] \leftarrow r'$. If the extraction fails, abort. Both games proceed identical unless the abort event happens. This only happens with negligible probability due to extractability property of the proof system. For simplicity both extractions are collapsed in a single game change. It is easy to unroll them into two separate game changes.

Now, if the adversary $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ such that $h^* = p \cdot e(g_1^a, h') = p' \cdot e(g_1^{a'}, h')$ in Game 7, $\mathcal{B}$ proceeds as follows. By definition, it holds that $g_T^{rx+ats} = g_T^{r'x+a'ts}$ (Note, $g^{ts} = h'$ in both cases, which, by definition, needs to be returned by the $\mathsf{Hash}_{\mathsf{CHET}}$ oracle, and thus $s = \mathtt{S}[h']$ is known). It follows that $rx + ats = r'x + a'ts$ is true. As all variables but t are now known (the values for r and r' can be obtained from $\mathtt{R}$), it holds that t can be calculated and returned as DL solution unless $a = a'$, which would however imply a collision for the hash function. All intermediate game changes are negligible, which concludes the proof. □

Note, IND-CCA2-security is required in the construction, as the adaption algorithm acts as a decryption oracle. Moreover, the construction does not achieve uniqueness, as the holder of $\mathsf{sk}_{\mathsf{CH}}$ can always derive new randomness, e.g., by re-encrypting r_1. This also true for the first construction in known-order groups.

6.4 A Small Note on Chameleon-Signatures

Chameleon-Signatures have been introduced by Krawczyk and Rabin at NDSS '00 [KR00b]. In a nutshell, the sender hashes the message m to be signed using the public key $\mathsf{pk}_{\mathsf{CH}}$ of a chameleon-hash with fresh randomness r. The resulting hash is then signed using a standard digital signature scheme. The randomness r, the signature σ, and the message m is then transferred to the receiver using a secure channel. As the receiver has the corresponding secret key $\mathsf{sk}_{\mathsf{CH}}$ of the chameleon-hash, it can create arbitrary collisions. Thus, following the reasoning of Krawczyk and Rabin [KR00b], the signature becomes "non-transferable" in the sense that the intended recipient cannot present the received signature to another third party, as the third party cannot decide whether the recipient used its secret key $\mathsf{sk}_{\mathsf{CH}}$ to create a collision, i.e., created a new signature on a new message m'. Hence, their construction inherently requires that a PKI vouches for the fact that the recipient actually knows $\mathsf{sk}_{\mathsf{CH}}$ (which they state), and thus does not choose it in way that it can prove to a third party that it does not know $\mathsf{sk}_{\mathsf{CH}}$, e.g., by letting $\mathsf{pk}_{\mathsf{CH}}$ be defined as the output of a random oracle. This represents, to a certain point, a "rogue key"-attack [RY07].

From a practical perspective, this can be solved very easily. In particular, the owner of the public key needs to attach a proof π, e.g., a NIZKPoK, that is knows the corresponding secret

key. This also removes, depending on the NIZKPoK, the need for a *trusted* third party, ignoring some potential setup assumptions, e.g., a common reference string or the like. How these proofs can be exactly be realized depends on the instantiation of the chameleon-hash in question and is not part of this note.

6.5 Application: Invisible Sanitizable Signatures

Informally, security of digital signatures requires that a signature σ on a message m becomes invalid as soon as a single bit of m is altered [GMR88]. However, there are many real-life use-cases in which a subsequent change to signed data by a semi-trusted party without invalidating the signature is desired. As a simplified example, consider a patient record which is signed by a medical doctor. The accountant, which charges the insurance company, only requires knowledge of the treatments and the patient's insurance number. This protects the patient's privacy. In this constellation, having the data re-signed by the M.D. whenever subsets of the record need to be forwarded to some party induces too much overhead to be practical in real scenarios or may even be impossible due to availability constraints.

Sanitizable signature schemes (SSS) [ACdMT05] address these shortcomings. They allow the signer to determine which blocks $m[i]$ of a given message $m = (m[1], m[2], \ldots, m[i], \ldots, m[\ell])$ are admissible. Any such admissible block can be changed to a different bitstring $m[i]' \in \{0,1\}^*$, where $i \in [1, \ell]$, by a semi-trusted party named the sanitizer. This party is identified by a private/public key pair and the sanitization process described before requires the private key. In a nutshell, sanitization of a message m results in an altered message $m' = (m[1]', m[2]', \ldots, m[i]', \ldots, m[\ell]')$, where $m[i] = m[i]'$ for every non-admissible block, and also a signature σ', which verifies under the original public key. Thus, authenticity of the message is still ensured. In the prior example, for the server storing the data it is possible to already black-out the sensitive parts of a signed document without any additional communication with the M.D. and in particular without access to the signing key of the M.D.

Real-world applications of SSSs include the already mentioned privacy-preserving handling of patient data, secure routing, privacy-preserving document disclosure, credentials, and blank signatures [ACdMT05, BFLS10, BPS12, BPS13, CL13, DHS14, DHS14].

6.5.1 Contribution

This section introduces the notion of *invisible* SSSs. This strong privacy notion requires that a third party not holding any secret keys cannot decide whether a specific block is admissible, i.e., can be sanitized. This has already been discussed by Ateniese et al. [ACdMT05] in the first work on sanitizable signatures, but they neither provide a formal framework nor a provably secure construction. However, some use-cases are identified where such a notion is important, and this gap is closed by introducing a new framework for SSSs, along with an extended security model. Moreover, a construction is proposed, which is provably secure in the new framework. The construction paradigm is based on IND-CPA secure encryption schemes, standard, yet unique, chameleon-hashes and strongly unforgeable signature schemes. These can be considered standard tools nowadays. Those are paired with a chameleon-hash with ephemeral trapdoors.

6.5.2 Motivation

At PKC '09, Brzuska et al. formalized the most common security model of SSSs [BFF+09]. For this work, the most important property they are addressing is "weak transparency". It means that although a third party sees which blocks of a message are admissible, it cannot decide whether some block has already been sanitized by a sanitizer. More precisely, their formalization explicitly requires that the third party is always able to decide whether a given block in a message is admissible. However, as this may invade privacy, having a construction which hides this additional information is useful as well. To address this problem the notion of "strong transparency" has been informally proposed in the original work by Ateniese et al. [ACdMT05].

Examples. To make the usefulness of such a stronger privacy property more visible, consider the following two application scenarios.

In the first scenario, consider that a document is the output of a workflow that requires several—potentially heavy—computations to become ready. It is assumed that the output of each workflow step could be produced by one party alone, but could also be outsourced. However, if the party decides to outsource the production of certain parts of the document it wants the potential involvement of other parties to stay hidden, e.g., the potential and actual outsourcing might be considered a trade secret. In order to regain some control that all tasks are done only by authorized subordinates, the document—containing template parts—is signed with a sanitizable signature. Such an approach, i.e., to use SSS for workflow control, was proposed by Derler et al. [DHPS15].

The second one is motivated by an ongoing legal debate in Germany.[8] Consider a school class where a pupil suffers from dyslexia[9] and thus can apply for additional help to compensate the illness. One way to compensate this is to consider spelling mistakes less when giving grades. Assume that only the school's principal shall decide to what extent a certain grade shall be improved. Of course, this shall only be possible for pupils who are actually handicapped. For the pupil with dyslexia, e.g., known to the teacher of the class in question, the grade is marked as sanitizable by the principal. The legal debate in Germany is about an outsider, e.g., future employer, who should not be able to decide that grades had the potential to be altered and of course also not see for which pupils the grades have been altered to preserve their privacy. To achieve this, standard sanitizable signature schemes are clearly not enough, as they do not guarantee that an outsider cannot derive which blocks are potentially sanitizable, i.e., which pupil is actually handicapped. The presented primitive offers a solution to this challenge, where an outsider cannot decide which block is admissible, i.e., can be altered.

6.5.3 State-of-the-Art

SSSs have been introduced by Ateniese et al. [ACdMT05]. Brzuska et al. formalized most of the current security properties [BFF+09]. These have been later extended for (strong) unlinkability [BFLS10, BPS13, FKM+16] and non-interactive public accountability [BPS12, BPS13]. Some properties discussed by Brzuska et al. [BFF+09] have then been refined by Gong

[8] See for example the ruling from the German Federal Administrative Court (BVerwG) 29.07.2015, Az.: 6 C 33.14, 6 C 35.14.

[9] A disorder involving difficulty in learning to read or interpret words, letters and other symbols.

et al. [GQZ10]. Namely, they also consider the admissible blocks in the security games, while still requiring that these are visible to everyone. Recently, Krenn et al. further refined the security properties to also account for the signatures, not only the message [KSS15].[10] The aforementioned results are used as the starting point for the extended definitions introduced in this chapter, as they cover most use-cases.

Also, several extensions such as limiting the sanitizer to signer-chosen values [CJ10, DS15, KL06, PSP11], trapdoor SSSs (which allow to add new sanitizers after signature generation by the signer) [CLM08, YSL10], multi-sanitizer and -signer environments for SSSs [BFLS09, BPS13, CJL12], and sanitization of signed and encrypted data [FF15] have been considered. SSSs have also been used as a tool to make other primitives accountable [PS15], and to build other primitives [BHPS16, dMPPS14]. Also, SSSs and data-structures being more complex than lists have been considered [PSP11]. Our results carry over to the aforementioned extended settings with only minor additional adjustments. Implementations of SSSs have also been presented [BPS12, BPS13, dMPPS13, PPS+13].

Of course, computing on signed messages is a broad field. This introduction can therefore only give a small overview. Decent and comprehensive overviews of other related primitives, however, have already been published [ABC+15, BPS17, BBD+10, DDH+15, GGOT16, GOT15, TDB16]. It is stressed, howeverm that this list is, by far, not exhaustive.

6.5.4 The Framework for Sanitizable Signature Schemes

Subsequently, the framework for SSSs is introduced. The definitions are based on existing work [BFF+09, BPS12, BPS13, GQZ10, KSS15]. However, due to the new goals, they were modifed to account for the fact that the admissible blocks are only visible to the sanitizer. Moreover, the property of "non-interactive public accountability" [BPS12, BPS13, HPS12] is not considered, which allows a third party to decide which party is accountable, as transparency is mutually exclusive to this property, but is elegantly to achieve, e.g., by signing the sanitizable signature again [BPS12]. For the sake of completeness, the definitions which are omitted in the following are provided in Appendix J.

Before the formal definition are presented, some additional notation is required. The variable ADM contains the set of indices of the modifiable blocks, as well as the number ℓ of blocks in a message m. Moreover, $\mathsf{ADM}(m) = \mathsf{true}$, if ADM is valid w.r.t. m, i.e., ADM contains the correct ℓ and all indices are in m. For example, let $\mathsf{ADM} = (\{1, 2, 4\}, 4)$. Then, m must contain four blocks, while all but the third will be admissible. The notation $m_i \in \mathsf{ADM}$ means that m_i is admissible. MOD is a set containing pairs $(i, m[i]')$ for those blocks that shall be modified, meaning that $m[i]$ is replaced with $m[i]'$. Likewise, the notation $\mathsf{MOD}(\mathsf{ADM}) = \mathsf{true}$ means that MOD is valid w.r.t. ADM, meaning that the indices to be modified are contained in ADM. To allow a compact presentation of the construction, let $\widetilde{X}_{n,m}$ with $n \leq m$ be the vector $(X_n, X_{n+1}, X_{n+2}, \ldots, X_{m-1}, X_m)$.

Definition 6.25 (Sanitizable Signatures)**.** *A sanitizable signature scheme* SSS *consists of the following eight PPT algorithms* ($\mathsf{PPGen}_{\mathsf{SSS}}, \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{Sign}_{\mathsf{SSS}}, \mathsf{Sanit}_{\mathsf{SSS}}, \mathsf{Verify}_{\mathsf{SSS}}, \mathsf{Proof}_{\mathsf{SSS}}, \mathsf{Judge}_{\mathsf{SSS}}$) *such that*

[10] Note, Krenn et al. [KSS15] also introduce "strong transparency", which is not related to the definition given by Ateniese et al. [ACdMT05].

$\mathsf{PPGen}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{PPGen}_{\mathsf{SSS}}$*, on input security parameter* λ*, generates the public parameters:*

$$\mathsf{pp}_{\mathsf{SSS}} \leftarrow \mathsf{PPGen}_{\mathsf{SSS}}(1^{\lambda})$$

It is assumed that $\mathsf{pp}_{\mathsf{SSS}}$ *is implicitly input to all other algorithms.*

$\mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}$ *takes the public parameters* $\mathsf{pp}_{\mathsf{SSS}}$ *and returns the signer's private key and the corresponding public key:*

$$(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$$

$\mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}$ *takes the public parameters* $\mathsf{pp}_{\mathsf{SSS}}$ *and returns the sanitizer's private key and the corresponding public key:*

$$(\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$$

$\mathsf{Sign}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{Sign}_{\mathsf{SSS}}$ *takes as input a message* m*,* $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}$*,* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}$*, as well as a description* ADM *of the admissible blocks. If* $\mathsf{ADM}(m) = \mathsf{false}$*, this algorithm returns* $\perp$*. It outputs a signature*

$$\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{SSS}}(m, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{ADM})$$

$\mathsf{Sanit}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{Sanit}_{\mathsf{SSS}}$ *takes a message* m*, modification instruction* MOD*, a signature* σ*,* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$ *and* $\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}$*. It outputs* m' *together with* σ'*:*

$$(m', \sigma') \leftarrow \mathsf{Sanit}_{\mathsf{SSS}}(m, \mathsf{MOD}, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})$$

Here, $m' \leftarrow \mathsf{MOD}(m)$ *is message* m *modified according to the modification instruction* MOD*.*

$\mathsf{Verify}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{Verify}_{\mathsf{SSS}}$ *takes as input the signature* σ *for a message* m *w.r.t. the public keys* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$ *and* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}$*. It outputs a decision* $d \in \{\mathsf{true}, \mathsf{false}\}$*:*

$$d \leftarrow \mathsf{Verify}_{\mathsf{SSS}}(m, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$$

$\mathsf{Proof}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{Proof}_{\mathsf{SSS}}$ *takes as input* $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}$*, a message* m*, a signature* σ*, a set of polynomially many additional message/signature pairs* $\{(m_i, \sigma_i)\}$*, and* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}$*. It outputs a string* $\pi \in \{0,1\}^*$ *which can be used by the* $\mathsf{Judge}_{\mathsf{SSS}}$ *to decide which party is accountable given a message/signature pair* (m, σ)*:*

$$\pi \leftarrow \mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$$

$\mathsf{Judge}_{\mathsf{SSS}}$. *The algorithm* $\mathsf{Judge}_{\mathsf{SSS}}$ *takes as input a message* m*, a signature* σ*,* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$*,* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}$*, as well as a proof* π*. Note, this means that once a proof* π *is generated, the accountable party can be derived by anyone for that message/signature pair* (m, σ)*. It outputs a decision* $d \in \{\mathsf{Signer}, \mathsf{Sanitizer}\}$*, indicating whether the message/signature pair has been created by the signer, or the sanitizer:*

$$d \leftarrow \mathsf{Judge}_{\mathsf{SSS}}(m, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \pi)$$

Experiment $\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \cdot, \cdot)}, \mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}, \mathcal{O}^{\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \cdot,\cdot,\cdot,\cdot)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
  for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i}, \mathsf{ADM}_i)$ and σ_i
    index the queries/answers to/from $\mathsf{Sign}_{\mathsf{SSS}}$
  for $j = 1, 2, \ldots, q'$ let $(m_j, \sigma_j, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},j}, \mathsf{MOD}_j)$ and (m'_j, σ'_j)
    index the queries/answers to/from $\mathsf{Sanit}_{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) = \mathsf{true} \wedge$
  $\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, m^*, \sigma^*) \neq (\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i}, m_i, \sigma_i) \wedge$
  $\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, m^*, \sigma^*) \neq (\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},j}, m'_j, \sigma'_j)$
return 0

Figure 6.8: SSS Unforgeability

Correctness of Sanitizable Signature Schemes. The usual correctness requirements must hold. In a nutshell, every signed and sanitized message/signature pair should verify, while a honestly generated proof on a honestly generated message/signature pair should point to the correct accountable party. A formal definition was given by Brzuska et al. [BFF+09], which straightforwardly extends to this framework.

6.5.5 Security of Sanitizable Signature Schemes

Next, the formal security model is introduced, where our definitions already incorporate newer insights [BFF+09, BPS13, GQZ10, KSS15]. In particular, mostly the "strong" definitions by Krenn et al. [KSS15] are considered as the new state-of-the-art, as they also capture malleability of the signatures. Moreover, also the data-structure corresponding to the admissible blocks, i.e., ADM, is an asset which needs protection, which addresses the work done by Gong et al. [GQZ10].

6.5.5.1 Unforgeability

The first notion introduced is unforgeability. This definition requires that an adversary $\mathcal{A}$ not having any secret keys is not able to produce any validating signature σ^* which it has not seen, even if $\mathcal{A}$ has full oracle access.

Definition 6.26 (Unforgeability)**.** *An* SSS *is unforgeable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure 6.8.

Experiment $\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^*, \mathsf{pk}^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot)}, \mathcal{O}^{\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot,\cdot,\cdot)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i}, \mathsf{ADM}_i)$ index the queries to $\mathsf{Sign}_{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^*) = \mathsf{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^* \neq \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i} \vee$
$m^* \notin \{\mathsf{MOD}(m_i) \mid \mathsf{MOD} \text{ with } \mathsf{ADM}_i(\mathsf{MOD}) = 1\})$
return 0

Figure 6.9: SSS Immutability

6.5.5.2 Immutability

Clearly, a sanitizer should only be able to sanitize the admissible blocks defined by ADM. This therefore also prohibits deleting or appending blocks from a given message. Moreover, the adversary is given full oracle access, while it is also allowed to generate the sanitizer key pair itself. Note, it is not required that the adversary cannot find any new signatures, as finding new signatures is clearly required due to the correctness of any SSS.

Definition 6.27 (Immutability)**.** *An* SSS *is immutable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure 6.9.

6.5.5.3 Privacy

The notion of privacy is related to the indistinguishability of ciphertexts. The adversary is allowed to input two messages with the same ADM which are sanitized to the exact same message. Afterwards, the adversary has to decide which message (left or right) was used to generate the sanitized one. The adversary receives full adaptive oracle access.

Definition 6.28 (Privacy)**.** *An* SSS *is private, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\left|\Pr[\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure 6.10.

6.5.5.4 Transparency

Transparency guarantees that the accountable party of a message m remains anonymous. This is important if discrimination may follow [ACdMT05, BFF+09]. In a nutshell, the adversary

Experiment $\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \xleftarrow{\$} \{0,1\}$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot)}, \mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}, \mathcal{O}^{\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot,\cdot,\cdot)}, \mathcal{O}^{\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}},b)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
where oracle $\mathsf{LoRSanit}$ on input of $m_0, \mathsf{MOD}_0, m_1, \mathsf{MOD}_1, \mathsf{ADM}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}$ and b
return $\perp$, if $\mathsf{MOD}_0(m_0) \neq \mathsf{MOD}_1(m_1) \ \vee \ \mathsf{ADM}(m_0) \neq \mathsf{ADM}(m_1)$
let $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{SSS}}(m_b, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{ADM})$
return $(m', \sigma') \xleftarrow{\$} \mathsf{Sanit}_{\mathsf{SSS}}(m_b, \mathsf{MOD}_b, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})$
return 1, if $a = b$
return 0

Figure 6.10: SSS Privacy

has to decide whether it sees a freshly signed signature, or a sanitized one. The adversary has full (but proof-restricted) adaptive oracle access. The proof-restriction is necessary to avoid trivial attacks. Moreover, the definition was adjusted to account for some subtleties regarding the restrictions of the proof oracle, in the sense of Bellare et al. for IND-CCA2 security [BHK15]. This has also been mentioned in a non-formal way by Derler and Slamanig [DS15].

Definition 6.29 ((Proof-Restricted) Transparency)**.** *An* SSS *is proof-restricted transparent, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure 6.11.

For the rest of the paper, "transparency" is written, even if proof-restricted transparency is meant. Moreover, our construction actually achieves the stronger notion of transparency, with the same arguments given by Brzuska et al. [BFLS10]. However, the proof-restricted version seems to be more natural.

6.5.5.5 Signer-Accountability

For signer-accountability, a signer should not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message/signature pair never issued by the sanitizer. Hence, the adversary $\mathcal{A}$ has to generate a proof π^* which makes $\mathsf{Judge}_{\mathsf{SSS}}$ to decide that the sanitizer is accountable, if it is not for a message m^* output by $\mathcal{A}$. Here, the adversary gains access to all oracles related to sanitizing. Note, this definition does not take the signature into account.

Definition 6.30 (Signer-Accountability)**.** *An* SSS *is signer-accountable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{Sig\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

Experiment $\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \xleftarrow{\$} \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot)}, \mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}, \mathcal{O}^{\mathsf{Proof}'_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot,\cdot,\cdot)}, \mathcal{O}^{\mathsf{SigOrSan}(\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}},b)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
where oracle $\mathsf{Proof}_{\mathsf{SSS}}'$ on input of $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$ and $\mathsf{pk}^{\mathsf{San}\prime}_{\mathsf{SSS}}$:
return $\perp$, if $\mathsf{pk}^{\mathsf{San}\prime}_{\mathsf{SSS}} = \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}} \ \wedge \ ((m,\sigma) \in \mathcal{Q} \ \vee \ \mathcal{Q} \cap \{(m_i,\sigma_i)\} \neq \emptyset)$
return $\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, m, \sigma, \{(m_i,\sigma_i)\}, \mathsf{pk}^{\mathsf{San}\prime}_{\mathsf{SSS}})$
where oracle $\mathsf{SigOrSan}$ on input of $m, \mathsf{MOD}, \mathsf{ADM}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}$ and b:
$\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathsf{SSS}}(m, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{ADM})$
$(m', \sigma') \xleftarrow{\$} \mathsf{Sanit}_{\mathsf{SSS}}(m, \mathsf{MOD}, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})$
if $b = 1$:
$\sigma' \xleftarrow{\$} \mathsf{Sign}_{\mathsf{SSS}}(m', \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{ADM})$
If $\sigma' \neq \perp$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma')\}$
return (m', σ')
return 1, if $a = b$
return 0

Figure 6.11: SSS (Proof-Restricted) Transparency

Experiment $\mathsf{Sig\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{pk}^*, \pi^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}}(\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
for $i = 1, 2, \ldots, q$ let (m'_i, σ'_i) and $(m_i, \mathsf{MOD}_i, \sigma_i, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},i})$
index the answers/queries from/to $\mathsf{Sanit}_{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) = \mathsf{true} \ \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*) \neq (\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},i}, m'_i) \ \wedge$
$\mathsf{Judge}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \pi^*) = \mathsf{Sanitizer}$
return 0

Figure 6.12: SSS Signer Accountability

where the experiment is defined in Figure 6.12.

Note, the adversary generates the signer public-key by itself and thus only receives access to the oracle related to sanitizing, but needs to return the generated public key and the forged proof.

Experiment $\mathsf{San\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^*, \mathsf{pk}^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \cdot, \cdot)}, \mathcal{O}^{\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \cdot, \cdot, \cdot, \cdot)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{ADM}_i, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i})$ and σ_i index the queries/answers to/from $\mathsf{Sign}_{\mathsf{SSS}}$
$\pi \xleftarrow{\$} \mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, \mathsf{pk}^*)$
return 1, if $\mathsf{Verify}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^*) = \mathsf{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i}, m_i, \sigma_i) \wedge$
$\mathsf{Judge}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^*, \pi) = \mathsf{Signer}$
return 0

Figure 6.13: SSS Sanitizer Accountability

6.5.5.6 Sanitizer-Accountability

Sanitizer-accountability requires that the sanitizer cannot blame the signer for a particular message/signature pair not created by the signer. In particular, the adversary has to make $\mathsf{Proof}_{\mathsf{SSS}}$ generate a proof π which makes $\mathsf{Judge}_{\mathsf{SSS}}$ decide that for a given (m^*, σ^*) generated by $\mathcal{A}$ the signer is accountable, while it is not. Thus, the adversary $\mathcal{A}$ gains access to all signer-related oracles.

Definition 6.31 (Sanitizer-Accountability)**.** *An* SSS *is sanitizer-accountable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\Pr[\mathsf{San\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

where the experiment is defined in Figure 6.13. Note, the adversary generates the sanitizer public-key by itself and receives full adaptive access to all signing-related oracles.

The notion of "non-interactive public accountability" [BPS12, BPS13, HPS12], which allows a third party to decide which party is accountable, is not considered, as transparency is mutually exclusive to this property, but is very easy to achieve, e.g., by signing the sanitizable signature again [BPS12]. The notion of unlinkability [BFLS10, BPS13, FKM+16, LZCS16], is not considered, as it seems to be hard to achieve with the underlying construction paradigm.

For completeness, these notions are given in Appendix J.

6.5.6 Invisibility of SSSs

Next, the new property of invisibility is introduced. Basically, invisibility requires that an outsider cannot decide which blocks of a given message are admissible. The notation $\mathsf{ADM}_0 \cap \mathsf{ADM}_1$ means the intersection of the admissible blocks, ignoring the length of the messages.

In a nutshell, the adversary can query an LoRADM oracle which either makes ADM_0 or ADM_1 admissible in the final signature. Of course, the adversary has to be restricted to $\mathsf{ADM}_0 \cap \mathsf{ADM}_1$

Experiment $\mathsf{Invisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \leftarrow \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \leftarrow \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sanit}'_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}},\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}, \mathcal{O}^{\mathsf{Proof}_{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot,\cdot,\cdot)}, \mathcal{O}^{\mathsf{LoRADM}(\cdot,\cdot,\cdot,\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}},\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},b)}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
where oracle LoRADM on input of $m, \mathsf{ADM}_0, \mathsf{ADM}_1, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}$ and b:
return $\bot$, if $\mathsf{ADM}_0(m) \neq \mathsf{ADM}_1(m)$
let $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SSS}}(m, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}\prime}_{\mathsf{SSS}}, \mathsf{ADM}_b)$
let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma, \mathsf{ADM}_0 \cap \mathsf{ADM}_1)\}$
return σ
where oracle $\mathsf{Sanit}'_{\mathsf{SSS}}$ on input of $m, \mathsf{MOD}, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$ and $\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}$:
return $\bot$, if $\nexists(m, \sigma, \mathsf{ADM}) \in \mathcal{Q} : \mathsf{MOD}(\mathsf{ADM}) = \mathsf{true}$
let $(m', \sigma') \leftarrow \mathsf{Sanit}_{\mathsf{SSS}}(m, \mathsf{MOD}, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})$
if $\exists(m, \sigma, \mathsf{ADM}') \in \mathcal{Q} : \mathsf{MOD}(\mathsf{ADM}') = \mathsf{true}$,
let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma', \mathsf{ADM}')\}$
return (m', σ')
return 1, if $a = b$
return 0

Figure 6.14: SSS Invisibility

for sanitization requests for signatures originating from those created by LoRADM and their derivatives to avoid trivial attacks. The sign oracle can be simulated by querying the LoRADM oracle with $\mathsf{ADM}_0 = \mathsf{ADM}_1$. It is stressed that the invisibility definition is very strong, as it also takes the signatures into account, much like the definitions given by Krenn et al. [KSS15]. One can easily alter the definition to only account for the messages in question, e.g., if one wants to avoid strongly unforgeable signatures, or even allow re-randomizable signatures. An adjustment is straightforward.

Definition 6.32 (Invisibility)**.** *An* SSS *is invisible, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\left|\Pr[\mathsf{Invisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure 6.14.

It is obvious that invisibility is not implied by any other property. In a nutshell, taking any secure SSS, it is sufficient to non-malleable append ADM to each block $m[i]$ to prevent invisibility. Clearly, all other properties of such a construction are still preserved.

Definition 6.33 (Secure SSS)**.** *An* SSS *is called secure, if it is correct, private, unforgeable, immutable, sanitizer-accountable, signer-accountable, and invisible.*

Note, neither non-interactive public accountability nor unlinkability nor transparency are considered as essential security requirements, as it depends on the concrete use-case whether these properties are required.

6.5.7 Construction

Now the construction, based on the paradigm of Ateniese et al. [ACdMT05], is introduced. It is enriched with several ideas of prior work [BFF+09, GQZ10, dMPPS13]. The main idea is to hash each block using a chameleon-hash with ephemeral trapdoors, and then sign the hashes. The main trick introduced to limit the sanitizer is that only those sktd_i are given to the sanitizer, for which the respective block $m[i]$ should be sanitizable. To hide whether a given block is sanitizable, each sktd_i is encrypted; a sanitizable block contains the real sktd_i, while a non-admissible block encrypts a 0, where 0 is assumed to be an invalid sktd. For simplicity, it is required that the IND-CPA secure encryption scheme ENC allows that each possible sktd, as well as 0, is in the message space $\mathcal{MS}$ of ENC, which can be achieved using standard embedding and padding techniques, or using KEM/DEM combinations [AGK08]. To achieve accountability, additional "tags" for a "standard" chameleon-hash (which binds everything together) are generated in a special way, namely PRFs and PRGs, which borrows ideas from the construction given by Brzuska et al. [BFF+09].

Construction 6.34 (Secure and Transparent SSS)**.** *The secure and transparent* SSS *construction is as follows:*

$\mathsf{PPGen}_{\mathsf{SSS}}$**.** *To generate the public parameters, do the following steps:*

1. *Let* $\mathsf{pp}_{\mathsf{CHET}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CHET}}(1^\lambda)$.
2. *Let* $\mathsf{pp}_{\mathsf{CH}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{CH}}(1^\lambda)$.
3. *Return* $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_{\mathsf{CHET}}, \mathsf{pp}_{\mathsf{CH}})$.

$\mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}$**.** *To generate the key pair for the signer, do the following steps:*

1. *Let* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.
2. *Pick a key for a* PRF*, i.e.,* $\kappa \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{PRF}}(1^\lambda)$.
3. *Return* $((\kappa, \mathsf{sk}_{\mathsf{Sig}}), \mathsf{pk}_{\mathsf{Sig}})$.

$\mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}$**.** *To generate the key pair for the sanitizer, do the following steps:*

1. *Let* $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$.
2. *Let* $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$.
3. *Let* $(\mathsf{sk}_{\mathsf{ENC}}, \mathsf{pk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$.
4. *Return* $((\mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{ENC}}), (\mathsf{pk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{ENC}}))$.

$\mathsf{Sign}_{\mathsf{SSS}}$**.** *To generate a signature* σ*, on input of* $m = (m[1], m[2], \ldots, m[\ell])$*,* $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}} = (\kappa, \mathsf{sk}_{\mathsf{Sig}})$*,* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}} = (\mathsf{pk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{ENC}})$*, and* ADM *do the following steps:*

1. *If* $\mathsf{ADM}(m) \neq \mathsf{true}$*, return* $\perp$.

2. *Draw* $x_0 \xleftarrow{\$} \{0,1\}^\lambda$.
3. *Let* $x_0' \leftarrow \mathsf{Eval}_{\mathsf{PRF}}(\kappa, x_0)$.
4. *Let* $\tau \leftarrow \mathsf{Eval}_{\mathsf{PRG}}(x_0')$.
5. *For each* $i \in [1,\ell]$ *do:*
   (a) *Set* $(h_i, r_i, \mathsf{sktd}_i) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, (i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}))$.
   (b) *If block* i *is not admissible, let* $\mathsf{sktd}_i \leftarrow 0$.
   (c) *Compute* $c_i \leftarrow \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sktd}_i)$.
6. *Set* $(h_0, r_0) \xleftarrow{\$} \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CHET}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}))$.
7. *Set* $\sigma' \xleftarrow{\$} \mathsf{Sign}_{\mathsf{Sig}}(\mathsf{sk}_{\mathsf{Sig}}, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \ell))$.
8. *Return* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$.

$\mathsf{Verify}_{\mathsf{SSS}}$. *To verify a signature* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$, *on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *w.r.t. to* $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}$ *and* $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}} = (\mathsf{pk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{ENC}})$, *do:*

1. *For each* $i \in [1,\ell]$ *do:*
   (a) *Set* $b_i \leftarrow \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, (i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i, h_i)$. *If any* $b_i = \mathsf{false}$, *return* false.
2. *Let* $b_0 \leftarrow \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_0, h_0)$.
3. *If* $b_0 = \mathsf{false}$, *return* false.
4. *Return* $d \leftarrow \mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \ell), \sigma')$.

$\mathsf{Sanit}_{\mathsf{SSS}}$. *To sanitize a signature* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$, *on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *w.r.t. to* $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}} s$, $\mathsf{sk}_{\mathsf{SSS}}^{\mathsf{San}} = (\mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{ENC}})$, *and* MOD *do:*

1. *Verify the signature, i.e., run* $d \leftarrow \mathsf{Verify}_{\mathsf{SSS}}(m, \sigma, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}})$. *If* $d = \mathsf{false}$, *return* $\perp$.
2. *Decrypt each* c_i *for* $i \in [1,\ell]$, *i.e., let* $\mathsf{sktd}_i \leftarrow \mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c_i)$. *If any decryption fails, return* $\perp$.
3. *For each index* $i \in \mathsf{MOD}$ *check that* $\mathsf{sktd}_i \neq 0$. *If not, return* $\perp$.
4. *For each block* $m[i]' \in \mathsf{MOD}$ *do:*
   (a) *Let* $r_i' \xleftarrow{\$} \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, (i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), (i, m[i]', \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i, h_i, \mathsf{sktd}_i)$.
   (b) *If* $r_i' = \perp$, *return* $\perp$.
5. *For each block* $m[i]' \notin \mathsf{MOD}$ *do:*
   (a) *Let* $r_i' \leftarrow r_i$.
6. *Let* $m' \leftarrow \mathsf{MOD}(m)$.
7. *Draw* $\tau' \leftarrow \{0,1\}^{2\lambda}$.
8. *Let* $r_0' \leftarrow \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), (0, m', \tau', \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r'}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_0, h_0)$.
9. *Return* $(m', (\sigma', x_0, \widetilde{r'}_{0,\ell}, \tau', \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell}))$.

$\mathsf{Proof}_{\mathsf{SSS}}$. *To create a proof* π, *on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *a signature* σ, *w.r.t. to* $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}$ *and* $\mathsf{sk}_{\mathsf{SSS}}^{\mathsf{Sig}}$, *and* $\{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$ *do:*

1. *Return* $\perp$, *if* $\mathsf{false} = \mathsf{Verify}_{\mathsf{SSS}}(m, \sigma, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}})$.

2. *Verify each signature in the list, i.e., run* $d_i \leftarrow \mathsf{Verify}_{\mathsf{SSS}}(m_i, \sigma_i, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$. *If for any* $d_i = \mathsf{false}$, *return* $\perp$.
3. *Go through the list of* (m_i, σ_i) *and find a (non-trivial) colliding tuple of the chameleon-hash with* (m, σ), *i.e.,* $h_0 = h'_0$, *where also* $\mathsf{true} = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r_0, h_0)$, *and* $\mathsf{true} = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, (0, m', \tau', \ell, \widetilde{h'}_{1,\ell}, \widetilde{c'}_{1,\ell}, \widetilde{r'}_{1,\ell}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r'_0, h'_0)$ *for some different tag* τ' *or message* m'. *Let this signature/message pair be* $(\sigma', m') \in \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$.
4. *Return* $\pi = ((\sigma', m'), \mathsf{Eval}_{\mathsf{PRF}}(\kappa, x_0))$, *where* x_0 *is contained in* (σ, m).

$\mathsf{Judge}_{\mathsf{SSS}}$. *To find the accountable party on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *a valid signature* σ, *w.r.t. to* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}$, $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$, *and a proof* π *do:*

1. *Check if* π *is of the form* $((\sigma', m'), v)$ *with* $v \in \{0,1\}^\lambda$. *If not, return* Signer.
2. *Also return* $\perp$, *if* $\mathsf{false} = \mathsf{Verify}_{\mathsf{SSS}}(m', \sigma', \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$, *or* $\mathsf{false} = \mathsf{Verify}_{\mathsf{SSS}}(m, \sigma, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$.
3. *Let* $\tau'' \leftarrow \mathsf{Eval}_{\mathsf{PRG}}(v)$.
4. *If* $\tau' \neq \tau''$, *return* Signer.
5. *If we have a (non-trivial) collision* $h_0 = h'_0$, $\mathsf{true} = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r_0, h_0) = \mathsf{Check}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, (0, m', \tau', \ell', \widetilde{h'}_{1,\ell'}, \widetilde{c'}_{1,\ell'}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r'_0, h'_0)$, $\widetilde{c}_{1,\ell} = \widetilde{c'}_{1,\ell'}$, $x_0 = x'_0$, $\ell = \ell'$, *and* $\widetilde{h}_{0,\ell} = \widetilde{h'}_{0,\ell'}$, *return* Sanitizer.
6. *Return* Signer.

Theorem 6.35. *If* ENC *is IND-CPA secure,* DSIG, PRF, PRG, CHET *are secure,* CH *is secure and unique, Construction 6.34 is a secure and transparent* SSS.

Note, CHET is not required to be unique.
Each property is proven on its own.

Proof. Correctness follows by inspection.

Unforgeability. To prove that the scheme is unforgeable, a sequence of games is used:

Game 0: The original unforgeability game

Game 1: As GAME 0, but abort if the adversary outputs a forgery (m^*, σ^*) with $\sigma^* = (\sigma'^*, x_0^*, \widetilde{r}^*_{0,\ell^*}, \widetilde{\tau}^*, \widetilde{c}^*_{1,\ell^*}, \widetilde{h}^*_{0,\ell^*})$, where $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \ell))$ was never obtained from the sign or sanitizing oracle. Let this event be E_1. Clearly, if $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$. For every required "inner" signature σ', use the signing oracle provided by the challenger. Now, whenever E_1 happens, output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery. Further, both games proceed identically, unless E_1 happens.

Game 2: Among others, it is now established that the adversary can no longer win by modifying $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}$ and $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}$. Now proceed as in GAME 1, but abort if the adversary outputs a forgery (m^*, σ^*), where message m^* or any of the other values protected by the outer chameleon-hash were never returned by the signer or the sanitizer oracle. Let this event be E_2. The probability of the abort event E_2 to happen is exactly the probability of the adversary breaking collision freeness for the outer chameleon-hash. Namely, it is already established that the adversary cannot tamper with the inner signature and therefore the hash value h_0^* must be from a previous oracle query. Now, assume that one obtains $\mathsf{pk}'_{\mathsf{ch}}$ from a collision freeness challenger. If E_2 happens, there must be a previous oracle query with associated values $(0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$ and r_0 so that h_0^* is a valid hash with respect to some those values and r_0. Further, it also holds that $(0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}) \neq (0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$, and can thus output $((0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_0^*, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_0, h_0^*)$ as the collision. Thus, the probability that E_2 happens is exactly the probability of a collision for the chameleon-hash. Both games proceed identically, unless E_2 happens.

Game 3: As GAME 2, but abort if the adversary outputs a forgery where only the randomness r_0 changed, i.e., there is previously generated signature with respect to r_0 so that $r_0 \neq r_0^*$. Let this be event be E_3. If the abort event E_3 happens, the adversary breaks uniqueness of the chameleon-hash. In particular, one receives values $(0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$ in the forgery which also correspond to some previous query, but r_0 from the previous query is different from r_0^*. Obtaining $\mathsf{pp}_{\mathsf{CH}}$ from a uniqueness challenger thus shows that E_3 happens with exactly the same probability as the adversary breaks uniqueness of the chameleon hash.

In the last GAME, the adversary can no longer win the unforgeability GAME; this GAME is computationally indistinguishable from the original GAME, which concludes the proof.

Immutability. Now is proven that the above construction is immutable using a sequence of games.

Game 0: The immutability game.

Game 1: As GAME 0, but abort if the adversary outputs a forgery (m^*, σ^*) with $\sigma^* = (\sigma'^*, x_0^*, \widetilde{r}^*_{0,\ell^*}, \widetilde{\tau}^*, \widetilde{c}^*_{1,\ell^*}, \widetilde{h}^*_{0,\ell^*})$ where $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \ell))$ was never obtained from the sign oracle. Let E_1 be the abort event. Clearly, if $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}$. For every required "inner" signature σ', use the signing oracle provided by the challenger. Now, whenever E_1 happens, one can output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery of the underlying signature scheme. Further, both games proceed identically, unless E_1 happens.

Game 2: As GAME 1, but the challenger aborts, if the message m^* is not derivable from any returned signature. Note, it is already known that tampering with the signatures is not possible, and thus $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}$, and $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}}$, are fixed. The same is true for deleting or appending blocks, as ℓ is

signed in every case. Let this event be denoted E_2. Now assume that E_2 is non-negligible. One can then construct an adversary $\mathcal{B}$ which breaks the private collision-resistance of the underlying chameleon-hash with ephemeral trapdoors. Let the signature returned be $\sigma^* = (\sigma'^*, x_0^*, \widetilde{r}_{0,\ell^*}^*, \widetilde{\tau}^*, \widetilde{c}_{1,\ell^*}^*, \widetilde{h}_{0,\ell^*}^*)$, while $\mathcal{A}$'s public key is pk^*. Due to prior game hops, it is known that $\mathcal{A}$ cannot tamper with the "inner" signatures. Thus, there must exists another signature $\sigma = (\sigma'^*, x_0^*, \widetilde{r'}_{0,\ell^*}^*, \widetilde{\tau}'^*, \widetilde{c}_{1,\ell^*}^*, \widetilde{h}_{0,\ell^*}^*)$ returned by the signing oracle. This, however, also implies that there must exists an index $i \in [1, \ell^*]$, for which one has $\mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, (i, m^*[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i^*, h_i^*) = \mathsf{CheckHash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CH}}, (i, m'^*[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i'^*, h_i^*) = \mathsf{true}$, where $m^*[i] \neq m'^*[i]$ by assumption. $\mathcal{B}$ proceeds as follows. Let q_h be the number of "inner hashes" created. Draw an index $i \xleftarrow{\$} [1, q_h]$. For a query $i \neq j$, proceed as in the algorithms. If $i = j$, however, $\mathcal{B}$ returns the current public key pk_c for the chameleon-hash with ephemeral trapdoors. This key is contained in $\mathsf{pk}_{\mathsf{SSS}}^{\mathsf{San}*}$. $\mathcal{B}$ then receives back control, and queries its $\mathsf{Hash}_{\mathsf{CHET}}$ oracle with $(i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$, where i is the current index of the message m to be signed. Then, if $((i, m^*[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i^*, (i, m'^*[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}}), r_i'^*, h_i^*)$ is the collision w.r.t. pk_c, it can directly return it.

As each hop changes the view of the adversary only negligibly, immutability is proven, as the adversary has no other way to break immutability in Game 2.

Privacy. Now privacy is proven, again using a sequence of games.

Game 0: The original privacy game.

Game 1: As Game 0, but abort if the adversary queries a verifying message-signature pair (m^*, σ^*) which was never returned by the signer or the sanitizer oracle, and queries it to the sanitization or proof generation oracle. Let E_1 be the abort event. Clearly, whenever the adversary queries such a new pair, one can output it to break the unforgeability of the scheme, as this tuple is fresh. However, it was already proven that this can only happen with negligible probability.

Game 2: As Game 1, but instead of hashing the blocks $(i, m_b[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$ for the inner chameleon-hashes using $\mathsf{Hash}_{\mathsf{CHET}}$, and then $\mathsf{Adapt}_{\mathsf{CH}}$ to $(i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$, directly apply $\mathsf{Hash}_{\mathsf{CHET}}$ to $(i, m[i], \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$. Assume that the adversary can distinguish this hop. One can then construct an adversary $\mathcal{B}$ which wins the indistinguishability game $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{CH}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

Game 3: As Game 2, but instead of adapting $(0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{SSS}}^{\mathsf{Sig}})$ to the new values, directly use $\mathsf{Hash}_{\mathsf{CHET}}$. Assume that the adversary can distinguish this hop. One can then construct an $\mathcal{B}$ which wins the indistinguishability game $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{CH}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

Clearly, the game is now independent of the bit b. As each hop changes the view of the adversary only negligibly, privacy is proven.

Transparency. Now transparency is proven by showing that the distributions of sanitized and fresh signatures are indistinguishable. Note, the adversary is not allowed to query $\mathsf{Proof}_{\mathsf{SSS}}$ for values generated by $\mathsf{SigOrSan}$.

Game 0: The original transparency GAME, where $b = 0$.

Game 1: As GAME 0, but abort if the adversary queries a valid message-signature pair (m^*, σ^*) which was never returned by any of the calls to the sanitization or signature generation oracle. Let us use E_1 to refer to the abort event. Clearly, whenever the adversary queries such a new pair, one can output it to break the unforgeability of the scheme, as this tuple is fresh. A reduction is straightforward.

Game 2: As GAME 1, but instead of computing $x'_0 \leftarrow \mathsf{Eval}_{\mathsf{PRF}}(\lambda, x_0)$, set $x'_0 \leftarrow \{0,1\}^\lambda$ within every call to $\mathsf{Sign}_{\mathsf{SSS}}$ in the $\mathsf{SigOrSan}$ oracle. A distinguisher between these two games straightfowardly yields a distinguisher for the PRF.

Game 3: As GAME 2, but instead of computing $\tau \leftarrow \mathsf{Eval}_{\mathsf{PRG}}(x'_0)$, set $\tau \xleftarrow{\$} \{0,1\}^{2\lambda}$ for every call to $\mathsf{Sign}_{\mathsf{SSS}}$ within the $\mathsf{SigOrSan}$ oracle. A distinguisher between these two games yields a distinguisher for the PRG using a standard hybrid argument.

Game 4: As GAME 3, but abort if a tag τ was drawn twice. Let this event be E_4. As the tags τ are drawn completely random, event E_4 only happens with probability $\frac{q_t^2}{2^{2\lambda}}$, where q_t is the number of drawn tags.

Game 5: As GAME 4, but instead of hash and then adapting the inner chameleon-hashes, directly hash $(i, m[i], \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}})$. Assume that the adversary can distinguish this hop. One can then construct an $\mathcal{B}$ which wins the indistinguishability game In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{CH}}$, and proceeds honestly except that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

Game 6: As GAME 5, but instead of hashing and then adapting the outer hash, directly hash the message, i.e., $(0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}})$. Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, embeds pk_c as $\mathsf{pk}_{\mathsf{CH}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

Clearly, now $b = 1$ holds, while each hop changes the view of the adversary only negligibly. This concludes the proof.

Signer-Accountability. Now is proven that the construction is signer-accountable using a sequence of games.

Game 0: The original signer-accountability game

Game 1: As GAME 0, but abort if the sanitization oracle draws a tag τ' which is in the range of the PRG. Let this event be E_1. This hop is indistinguishable by a standard statistical argument: at most 2^λ values lie in the range of the PRG. Note, this also means, that there exists no valid pre-image x_0.

Game 2: As GAME 1, but now abort, if the adversary was able to find $(\mathsf{pk}^*, \pi^*, m^*, \sigma^*)$ for some message m^* with a τ^* which was never returned by the sanitization oracle. Let this event be E_2. The previous games have already established that the sanitizer oracle will never return a signature with respect to a tag τ in the range of the PRG. Thus, if event E_2 happens, one knows by the condition checked in step 4 of $\mathsf{Judge}_{\mathsf{SSS}}$ that at least one of the tags (either τ^* in σ^*, or τ^π in π^*) was chosen by the adversary, which, in further consequence, implies a collision for CH. Namely, assume that E_3 happens with non-negligible probability. Then embed the challenge public key pk_c in $\mathsf{pk}'_{\mathsf{CH}}$, and use the provided adaption oracle to simulate the sanitizer oracle. If E_3 happens one can output $((0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r^*_0, (0, m'^*, \tau'^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}_{1,\ell}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r'^*_0, h^*_0)$, as a valid collision. These values can simply be compiled using π^*, m^*, and σ^*.

Game 3: As GAME 2, but now abort, if the adversary was able to find $(\mathsf{pk}^*, \pi^*, m^*, \sigma^*)$ for a new message m^* which was never returned by the sanitization oracle. Let this event be E_3. Assume that E_3 happens with non-negligible probability. In the previous games it was already established that the only remaining possibility for the adversary is to re-use tags τ^*, τ^π corresponding to some query/response to the sanitizer oracle. Then, m^* must be fresh, as it was never returned by the sanitization oracle by assumption. Thus, $((0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r^*_0, (0, m'^*, \tau'^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}_{1,\ell}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r'^*_0, h^*_0)$, is a valid collision. These values can simply be compiled using π^*, m^*, and σ^*.

In the last GAME the adversary can no longer win; each hop only changes the view negligibly. This concludes the proof.

Sanitizer-Accountability. That the construction is sanitizer-accountable is proven by a sequence of games.

Game 0: The original sanitizer-accountability definition.

Game 1: As GAME 0, but abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{pk}^*)$ with $\sigma^* = (\sigma'^*, x^*_0, \widetilde{r}^*_{0,\ell^*}, \widetilde{\tau}^*, \widetilde{c}^*_{1,\ell^*}, \widetilde{h}^*_{0,\ell^*})$ where $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \ell))$ was never obtained from the signing oracle. Let the abort event be E_1. Clearly, if $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}$. For every required "inner" signature σ', use the signing oracle provided by the challenger. Now, whenever E_1 happens, one can output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery. Further, both games proceed identically, unless E_1 happens.

Game 2: As GAME 1, but abort if the adversary outputs a forgery where only the randomness r_0 changed, i.e., there is previously generated signature with respect to r_0 so that $r_0 \neq r_0^*$. Let this event be E_2. If the abort event E_2 happens, the adversary breaks uniqueness of the chameleon-hash. In particular, one has values $(0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}})$ in the forgery which also correspond to some previous query, but r_0 from the previous query is different from r_0^*. Obtaining $\mathsf{pp}_{\mathsf{CH}}$ from a uniqueness challenger thus shows that E_2 happens with exactly the same probability as the adversary breaks uniqueness of the chameleon hash.

In GAME 2 the forgery is different from any query/answer tuple obtained using $\mathsf{Sign}_{\mathsf{SSS}}$ by definition. Due to the previous hops, the only remaining possibility is a collision in the outer chameleon-hash, i.e., for $h_0^* = h_0'^*$ one has $\mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}'^*, (0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r_0^*, h_0^*) = \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}'^*, (0, m'^*, \tau'^*, \ell'^*, \widetilde{h}'^*_{1,\ell'^*}, \widetilde{c}'^*_{1,\ell'^*}, \widetilde{r}'^*_{1,\ell'^*}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}), r_0'^*, h_0'^*) = \mathsf{true}$. In this case the $\mathsf{Judge}_{\mathsf{SSS}}$ algorithm returns $\mathsf{Sanitizer}$ and $\Pr[S_2] = 0$ which concludes the proof.

Invisibility. Now is proven that the construction is invisible by a sequence of games. The idea is to show that one can simulate the view of the adversary without giving out any useful information at all.

Game 0: The original invisibility GAME, i.e., the challenger runs the experiment as defined.

Game 1: As GAME 0, but abort if the adversary queries a valid message-signature pair (m^*, σ^*) which was never returned by the signer or the sanitizer oracle to the sanitization or proof generation oracle. Let this abort event be E_1. Clearly, whenever the adversary outputs such a new pair, one can output it to break unforgeability of the scheme, as this tuple is fresh. However, it was already proven that this can only happen with negligible probability. Note, this also means that only those signatures can be input to the sanitization oracle which have both of the challenge keys signed.

Game 2: As GAME 1, but internally keep all sktd_i. This is only a conceptual change.

Game 3: As GAME 2, but encrypt only zeroes instead of the real sktd_i in LoRADM independent of whether block are admissible or not. Note, the LoRADM oracle enforces that $\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}} = \mathsf{pk}^{\mathsf{San}\prime}_{\mathsf{SSS}}$. Note, the challenger still knows all sktd_i, and can thus still sanitize correctly. A standard reduction, using hybrids, shows that this hop is indistinguishable by the $\mathsf{IND\text{-}CPA}$ security of the encryption scheme used. Note, $\mathsf{IND\text{-}CPA}$ security of the encryption scheme ENC is sufficient, as the abort in GAME 1 ensures that the adversary can only submit queries with respect to ciphertexts which were previously generated in the reduction, i.e., where one can simply look up the respective values sktd_i instead of decryption.

At this point, the distribution is independent of the LoRADM oracle. Note, the sanitization, and proof oracles, can be still be simulated without any restrictions, as each sktd_i is known to the challenger. Thus, the view the adversary receives is now completely independent of the bit b used in the invisibility definition. As each hop only changes the view of the adversary negligibly, the construction is thus proven to be invisible. □

6.6 Conclusion

This chapter has introduced the notion of chameleon-hashes with ephemeral trapdoors. This primitive allows to prevent the holder of the trapdoor corresponding to the long-term public key from finding collisions. Along with a comprehensive security model we have presented four provably secure constructions. The first one is bootstrapped from any chameleon-hash in a black-box fashion, while the second direct scheme is based on RSA-like assumptions, and the other two on the DL assumption. Applications of this new primitive include, but are not limited to, the first provably secure construction of invisible sanitizable signatures. An interesting open problem is the construction of sanitizable signatures which are invisible and unlinkable at the same time.

Chapter 7

Practical Signing-Right Revocation

The results of this chapter have already been published [BKPS16].

Abstract. One of the key features that must be supported by every modern PKI is an efficient way to determine (at verification) whether the signing key had been revoked. In most solutions, the verifier periodically contacts the certificate authority (CA) to obtain a list of blacklisted, or whitelisted, certificates. In the worst case this has to be done for every signature verification. Besides the computational costs of verification, after revocation *all* signatures under the revoked key become invalid. In the solution by Boneh et al. [BDTW01], the CA holds a share of the private signing key and contributes to the signature generation. After revocation, the CA simply denies its participation in the interactive signing protocol. Thus, the revoked user can no longer generate valid signatures. This solution is extended to also cover privacy, non-trusted setups, and time-stamps. This is accompanied by a formal definitional framework, and elegantly simple, yet provably secure, instantiations from efficient standard building blocks such as digital signatures, commitments, and partially blind signatures, which can co-exist with already deployed solutions. Finally, several extensions to the basic scheme are provided.

Roadmap. Section 7.1 presents the problem statement. The framework for CA-assisted signatures is presented in Section 7.2. The corresponding constructions are given in Section 7.3, including some extensions. The chapter is concluded in Section 7.4.

7.1 Introduction

Digital signatures [GMR88] provide meaningful security as long as the signing key stays secret. However, in the real-world, signing keys can be compromised very easily, e.g., through hacker attacks, lost hardware tokens, or simply by accident. Furthermore, it is often required to revoke signing rights, e.g., when an employee leaves a company. Consequently, deployed solutions such as X.509, and related standards, always allow for revocation of certificates [CDE+14, CSF+08]. Here, two main approaches (and potentially combinations thereof) are deployed. First, in a *white-list* approach, the certificate authority (CA) vouches for the fact that a given certificate is not revoked. Alternatively, the CA can publish a *black-list* containing all revoked certificates. Now, a verifier directly rejects a signature if the used key has been black-listed. Thus, if one

requires up-to-date information, this means that the lists must be retrieved *for every signature verification*, causing a high — and sometimes too high — computational and communicational overhead. Thus, in either case, the verifiers contact the CA to determine whether a given certificate is still valid. Hence, every verifier must periodically update the published lists in both approaches to have meaningful security guarantees.

Moreover, as noted by Boneh et al. [BDTW01], these *total* revocation mechanisms have several drawbacks. For example, as mentioned previously, to check the revocation status of a given certificate, the verifier must have access to an up-to-date certificate revocation list (CRL), or the CA has to be queried for each signature verification. The latter may not be possible, however, as the verifier may not have a network connection, or communication is too costly. Furthermore, if a certificate is revoked, all signatures corresponding to the contained public key pk, including the ones that were generated honestly, become invalid after revocation. However, it is desirable that all signatures under a secret key sk that were generated prior to the corruption of sk (or prior to the revocation of the corresponding certificate) remain valid, while the generation of new signatures under sk is not possible. For example, consider Spider-Man sending the message m="I admit that you, Iron Man, are more powerful than me."[1] Clearly, if m is signed with Spider-Man's secret key sk, Iron Man can publish the signature to prove to the public that he is more powerful than Spider-Man. However, if Spider-Man revokes his certificate, the signature becomes invalid, and there is no way for Iron Man to prove that the statement is valid. This is because if the secret key sk is corrupted, it cannot be proven that Iron Man is not the adversarial party generating *new* bogus signatures on behalf of Spider-Man. The problem is that signatures are not associated with their generation time, i.e., a new signature is as good as an old one, if no further means such as time-stamping services are involved. Thus, all signatures have to be revoked in this setting. Refer to Gutmann for additional problems of PKIs in their current form [Gut02].

7.1.1 Contribution

The aforementioned unsatisfactory situation is addressed by introducing the notion of CA-assisted signature generation with time-stamping, message privacy and non-trusted setup. In a nutshell, the scheme requires that a *partially* trusted CA blindly signs the message m in question plus potentially a time-stamp (and some other technical values such as keys, etc.), while a trusted setup is not required. In particular, the CA checks whether the corresponding user's pk is revoked and signs m only if pk not revoked. The signature generated by the CA is then additionally signed with a standard digital signature scheme by the user. Both signatures are subsequently sent to, and verified, by the verifier. Signatures can be generated as long as the corresponding public key is not revoked. Therefore, all generated signatures remain valid after revocation as the CA simply stops assisting the signer after the key gets revoked.

While technically being relatively simple, the construction solves most of the mentioned problems, and, interestingly enough, is even more efficient than most deployed solutions, as the CAs are no longer queried for each verification.

Moreover, it is required that the solution can be added "on-top" of the existing PKI, i.e., the users do not require new keys, while the existing method can co-exist. If a time-stamping

[1] For all Spider-Man fans: please reverse the roles of Spider-Man and Iron Man.

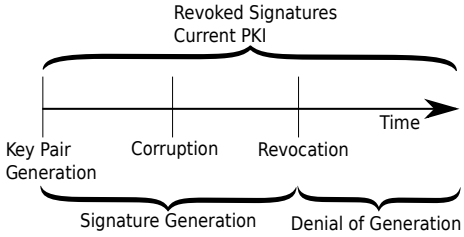

Figure 7.1: Revocation of certificates

authority and traditional revocation lists are naïvely used to solve the problem, the signing process needs to be interactive similar to the constructions given (because the time-stamp needs to be bound to the signed message). However, the solution does not require *any* interactivity upon verification, which is needed in the naïve solution in order to update revocation information. Moreover, the construction paradigm is elegantly simple, yet very versatile and black-box. It is shown how it can easily be extended to cover additional application scenarios. Interestingly, when one tries to close the remaining gap between corruption and revocation (refer to Figure 7.1), the resulting construction becomes very similar to the naïve solution again (Refer to Section 7.3.3). However, in this case it is easy to see that interactivity is needed for signing (because of the time-stamp), as well as for verification (to check whether a signature key has been revoked "into the past").

Even though the CA is only partially trusted, nothing is lost, as some kind of trust anchor is always required for a PKI anyway. The proposed approach actually requires less trust: for white-lists, the CA learns if signatures for a specific public key are verified, while in a black-list approach everyone sees which certificates are revoked. In the proposed solution, the CA only learns when a signature is generated, which happens less frequently. Moreover, the presented constructions offer a fall-back mode, which allows to revert to standard signatures.

A Word of Caution. Even though the algorithms involve a CA, a meaningful definition in practice requires that any certificates issued are generated by another entity. Otherwise, the security are meaningless in a practical sense, as the "CA" can always produce new certificates and bind a public key to a name. In other words, the CA in the protocols proposed here must be considered as "revocation authorities", and not CAs in the usual sense.

7.1.2 Related Work

The idea to let a (semi-)trusted entity such as a CA also contribute to signature generation has been introduced by Boneh et al. [BDTW01] and Rivest [Riv98], but neither present a full formalization. The approach by Boneh et al. is based on standard 2-out-of-2 threshold signatures [Bol03, DF89]. In particular, the secret key sk is split between the CA and the signer. The server denies its contribution to signature generation, if the presented certificate is marked as revoked. However, their approach requires trusted setup (the suggested mitigation strategy of using a distributed key generation algorithm here is too inefficient in practice), new keys for each participant and cannot add time-stamps to generated signatures. Moreover, an adversarial server may also learn the message to be signed, i.e., in contrast to the solution presented in this chapter, no privacy guarantees are given to the user. A similar approach is deployed in

anonymous credentials such as Idemix [CDE+14, CL02a], where the credential holder proves that it is not revoked at presentation of the credential, e.g., using accumulators [BdM93, BCD+17, CH02, DHS15, PS14]. Here, the prover has to prove knowledge of a witness (in zero-knowledge) such that its revocation handle is contained in the accumulator, which resembles a white-list approach. Clearly, the witnesses have to be updated for each revocation, while credentials are, compared to digital signatures, only valid once at presentation or, to be more precise, only valid for a certain amount of time to avoid re-usage.

There also exists the notion of certificate-less cryptography [AP03, HSMZ05]. Compared to the proposed construction, they also require a certificate, there are *ephemeral* keys and identity management. However, the ideas are very similar and can thus be seen as related. Likewise, the concept of *virtual smart-cards* (See Chapter 4) is related and actually serves as the basis for this chapter. However, in contrast to the proposed approach, the additional server is not trusted by outsiders and the signer has to provide an additional password. Moreover, for an outsider (i.e., verifier), a signature generated in such a scheme may be indistinguishable from a traditional signature. This is not what the proposed construction achieves, i.e., a verifier must be able to decide whether a signature was generated using the method presented.

The idea of using an additional entity to revoke public-keys has also been proposed in the setting of encryptions, e.g., for identity-based encryption [BGK08,DT03], for access-control [BDT04], but for also general encryption [CBN06]. In general, those schemes prohibit that a ciphertext can be decrypted. However, for every decryption (or key-update), the server has to be contacted, while in the constructions presented in this chapter, for each signature the server has to be asked once. Thus, the constructions presented compliment existing ideas for signatures, extending them for additional use-cases and add additional privacy features. If, and how, the ideas presented in this chapter can be extended to encryption is left as open work.

There are also other primitives which may be used to achieve similar goals, e.g., threshold signatures [DF89], proxy signatures [MUO96], server-assisted signatures [BB04], multi signatures [BN06], but also aggregate signatures [BGLS03], or sanitizable signatures [ACdMT05, BFF+09, KSS15].[2] However, all these approaches do not offer privacy (i.e., they potentially reveal the message to the server or signatures become linkable to a protocol execution) without further modifications. It was therefore chosen to use primitives which directly give the required security and privacy guarantees.

Blind signatures have been introduced by Chaum [Cha82]. In a nutshell, blind signatures allow an external entity to receive a signature σ on a message m (of its own choice) such that the signer learns nothing about the message m, and cannot link a signing transcript to the final signature. Chaum's work was later formalized and proven secure [BNPS03, JLO97]. Later, constructions in the standard model [CKW04], based on different assumptions other than RSA [Bol03], additional security guarantees [FS09], but also some impossibility results [FS10] were published.

The initial idea was also extended to cover some form of partial blindness, where the signature is issued on the blinded message m, but also some public information info known to both parties [AO00, CHYC05]. These partially blind signatures are mostly used to prevent misuse of blind signatures. This possibility is used in the proposed construction to bind a signature to a public key, but also to add time-stamps.

[2] See also Chapter 6.

7.1.3 Additional Preliminaries

For this chapter, the only additional building block required are partially blind signatures.

7.1.3.1 Partially Blind Signatures

Blind signatures [Cha82, JLO97] allow the holder of a secret key to sign a message m for a second entity. The signer does not learn what message it signs, and also cannot link a signature generation transcript against the final signature. Partially blind signatures [AO00] also allow to bind some piece of public information, known to both parties, to the final signature. Note, for the following definition, the case where some "public parameters" are generated is omitted, as it depends on the underlying scheme whether this algorithm is required. An extension is straightforward. In particular, one simply defines a new algorithm which outputs the public parameters, but one is also required to adjust the definitions to account for the new algorithm and the new values. However, for the use-case presented in this thesis, it is only of minor relevance.

Definition 7.1 (Partially Blind Signatures)**.** *A partially blind signature scheme* BSIG *consists of two algorithms* $\{\mathsf{KeyGen}_{\mathsf{BSIG}}, \mathsf{Verify}_{\mathsf{BSIG}}\}$*, and an interactive protocol* $\langle \mathcal{B}, \mathcal{U} \rangle$*, such that:*

$\mathsf{KeyGen}_{\mathsf{BSIG}}$**.** *The algorithm* $\mathsf{KeyGen}_{\mathsf{BSIG}}$ *outputs the public and private key of the signer, where* λ *is the security parameter:*

$$(\mathsf{sk}_{\mathsf{BSIG}}, \mathsf{pk}_{\mathsf{BSIG}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{BSIG}}(1^{\lambda})$$

$\langle \mathcal{B}, \mathcal{U} \rangle$**.** *The algorithm* $\langle \mathcal{B}, \mathcal{U} \rangle$ *is interactive. The user* $\mathcal{U}$ *receives input* m*, public information* info*, and* $\mathsf{pk}_{\mathsf{BSIG}}$*. The signer* $\mathcal{B}$ *inputs the secret key* $\mathsf{sk}_{\mathsf{BSIG}}$*, and some string* info*, while the user* $\mathcal{U}$ *inputs a public key* $\mathsf{pk}_{\mathsf{BSIG}}$*, a message* m*, and the string* info*. At the end of the protocol, only the user* $\mathcal{U}$ *receives a signature* σ*, while* $\mathcal{B}$ *receives nothing:*

$$(\perp, \sigma) \xleftarrow{\$} \langle \mathcal{B}(\mathsf{sk}_{\mathsf{BSIG}}, \mathsf{info}), \mathcal{U}(\mathsf{pk}_{\mathsf{BSIG}}, m, \mathsf{info}) \rangle$$

Here, $\langle \cdot, \mathcal{U}(\cdot,\cdot,\cdot) \rangle^{\infty}$ *means that if the adversary plays the role of the signer* $\mathcal{B}$*, it can start a new signing session with* $\mathcal{U}$ *as often as it wants to, and can arbitrarily schedule the interactions. Likewise, the notation* $\langle \mathcal{B}(\cdot,\cdot), \cdot \rangle^{1}$ *means, that the adversary acts as the user, and can interact with the signer only once. For simplicity it is also required that every entity is able to decide to what step of which "session" a given protocol message corresponds, and also when a given "signing session" is finished, and whether was successful. In particular, it is required that a signing session is finished once* $\mathcal{B}$ *sends its last message to* $\mathcal{U}$*, and* $\mathcal{U}$ *can actually extract a valid signature. This in particular means, that the* $\mathcal{B}$ *does not send any other message after* $\mathcal{U}$ *can extract a signature.*

$\mathsf{Verify}_{\mathsf{BSIG}}$**.** *The deterministic algorithm* $\mathsf{Verify}_{\mathsf{BSIG}}$ *outputs a decision bit* $d \in \{\mathsf{false}, \mathsf{true}\}$*, indicating the validness of the signature* σ*, w.r.t.* $\mathsf{pk}_{\mathsf{BSIG}}$*,* info*, and* m*:*

$$d \leftarrow \mathsf{Verify}_{\mathsf{BSIG}}(\mathsf{pk}_{\mathsf{BSIG}}, m, \mathsf{info}, \sigma)$$

Experiment $\mathsf{omUNF\text{-}CMA}^{\mathsf{BSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{BSIG}}, \mathsf{pk}_{\mathsf{BSIG}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{BSIG}}(1^\lambda)$
$((m_1, \sigma_1, \mathsf{info}_1), \ldots, (m_\ell, \sigma_\ell, \mathsf{info}_\ell)) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\langle \mathcal{B}(\mathsf{sk}_{\mathsf{BSIG}}, \cdot, \cdot), \cdot \rangle^\infty}}(\mathsf{pk}_{\mathsf{BSIG}})$
return 1, if $\forall i \in \{1, 2, \ldots, \ell\} : \mathsf{Verify}_{\mathsf{BSIG}}(\mathsf{pk}_{\mathsf{BSIG}}, m_i, \mathsf{info}_i, \sigma_i) = \mathsf{true}$
and oracle $\langle \mathcal{B}(\mathsf{sk}, \cdot), \cdot \rangle$ finished less than ℓ times, and
all (m_i, info_i) are pairwise distinct
return 0

Figure 7.2: BSIG Unforgeability

Correctness. For each BSIG it is required that the correctness properties hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $(\mathsf{sk}_{\mathsf{BSIG}}, \mathsf{pk}_{\mathsf{BSIG}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{BSIG}}(1^\lambda)$, for all $m \in \mathcal{MS}$, for all $\mathsf{info} \in \{0,1\}^*$, $\mathsf{Verify}_{\mathsf{BSIG}}(\mathsf{pk}_{\mathsf{BSIG}}, m, \mathsf{info}, \sigma) = \mathsf{true}$, where σ is taken from $(\perp, \sigma) \xleftarrow{\$} \langle \mathcal{B}(\mathsf{sk}_{\mathsf{BSIG}}, \mathsf{info}), \mathcal{U}(\mathsf{pk}_{\mathsf{BSIG}}, m, \mathsf{info}) \rangle$. This captures perfect correctness.

Security. Two different security notions are needed. These are presented next.

Unforgeability. The following unforgeability definition of partially blind signature schemes is based on the definition given by Abe and Okamoto [AO00, Oka06], but adjusted for the notation used in this thesis. In a nutshell, it is required that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with more signatures for different message/information pair (m, info) than successful, i.e., completed, signing queries. Note, the adversary can interleave signing queries.

Definition 7.2 (Unforgeability)**.** *A signature scheme* BSIG *is* ***unforgeable****, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathsf{omUNF\text{-}CMA}^{\mathsf{BSIG}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 7.2.

Note, this definition imposes "weak" unforgeability, i.e., once a signature for a given message/information pair (m, info) becomes known, the adversary may be able to derive new signatures for this tuple.

Blindness. Finally, blindness of partially blind signature schemes is defined, derived from the work by Okamoto [Oka06]. In a nutshell, it is required that an adversary $\mathcal{A}$ cannot (except with negligible probability) decide what message is signed, and cannot link a signing transcript against the final signature. This must even be true, if it can generate the public key, chose the messages to be signed, and also the public string info.

Definition 7.3 (Blindness)**.** *A partially blind signature scheme* BSIG *is* ***blind****, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathsf{Blindness}^{\mathsf{BSIG}}_{\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

Experiment $\mathsf{Blindness}^{\mathsf{BSIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}^*, \{m_0, m_1\}, \mathsf{info}, state_{\mathcal{A},1}) \xleftarrow{\$} \mathcal{A}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$state_{\mathcal{A},2} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\langle\cdot,\mathcal{U}_0(\mathsf{pk}^*,m_b,\mathsf{info})\rangle^1,\langle\cdot,\mathcal{U}_1(\mathsf{pk}^*,m_{1-b},\mathsf{info})\rangle^1}}(state_{\mathcal{A},1})$
let σ_0, and σ_1 denote the output of $\mathcal{U}_0$, and $\mathcal{U}_1$
If $\sigma_0 = \bot \vee \sigma_1 = \bot$, let $\sigma \leftarrow \bot$
Else, set $\sigma \leftarrow (\sigma_b, \sigma_{1-b})$
$a \xleftarrow{\$} \mathcal{A}(state_{\mathcal{A},2}, \sigma)$
return 1, if $a = b$
return 0

Figure 7.3: BSIG Blindness

The corresponding experiment is depicted in Figure 7.3.

Definition 7.4 (Secure BSIGs)**.** *A partially blind signature scheme* BSIG *is said to be secure, if it is correct, unforgeable, and blind.*

A concrete construction was given by Fuchsbauer et al. [FHKS16].

7.2 CA-Assisted Signatures

Now, CA-Assisted Signatures are introduced. As already discussed in the introduction, the main idea is that a CA helps generating a signature.

7.2.1 Framework for CA-Assisted Signatures

In the following a formal specification of the algorithms and their interfaces in such schemes are given. We require that each party has access to a common clock which is synchronized across all parties. In practice, this can be realized, e.g., by using the Network Time Protocol (NTP) [MMBK10], and checking that the time-stamp is in an acceptable range, say, e.g., 30 seconds.

Definition 7.5 (CA-Assisted Signatures)**.** *A CA-assisted digital signature scheme* CASIG *consists of four algorithms* $\{\mathsf{KeyGen}_\mathsf{U}, \mathsf{KeyGen}_\mathsf{CA}, \mathsf{Revoke}_\mathsf{CA}, \mathsf{Verify}_\mathsf{A}\}$ *and one interactive protocol* $\langle \mathcal{CA}, \mathcal{U} \rangle$ *such that:*

$\mathsf{KeyGen}_\mathsf{U}$**.** *The algorithm* $\mathsf{KeyGen}_\mathsf{U}$ *outputs the public and private key of each user, where* λ *is the security parameter:*

$$(\mathsf{sk}_\mathsf{User}, \mathsf{pk}_\mathsf{User}) \xleftarrow{\$} \mathsf{KeyGen}_\mathsf{U}(1^\lambda)$$

$\mathsf{KeyGen}_\mathsf{CA}$**.** *The algorithm* $\mathsf{KeyGen}_\mathsf{CA}$ *outputs the public and private key of a* $\mathcal{CA}$*, where* λ *is the security parameter:*

$$(\mathsf{sk}_\mathsf{CA}, \mathsf{pk}_\mathsf{CA}) \xleftarrow{\$} \mathsf{KeyGen}_\mathsf{CA}(1^\lambda)$$

$\langle \mathcal{CA}, \mathcal{U} \rangle$. *The protocol* $\langle \mathcal{CA}, \mathcal{U} \rangle$ *is interactive. The user* $\mathcal{U}$ *receives input* m, pk_s, time, *and* $\mathsf{sk}_{\mathsf{User}}$. *The* $\mathcal{CA}$ *inputs the secret key* $\mathsf{sk}_{\mathsf{CA}}$, time, *and* $\mathsf{pk}_{\mathsf{User}}$. *At the end of the protocol, only the user* $\mathcal{U}$ *receives a signature* σ *(which may be* $\perp$ *for a revoked user), while* $\mathcal{CA}$ *receives nothing:*

$$(\perp, \sigma) \xleftarrow{\$} \langle \mathcal{CA}(\mathsf{sk}_s, \mathsf{pk}_{\mathsf{User}}, \mathsf{time})), \mathcal{U}(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}, m, \mathsf{time}) \rangle$$

As for partially blind signatures, it is assumed that each party knows to which signing session, and which protocol step a received message belongs to, and is also successful.

$\mathsf{Revoke}_{\mathsf{CA}}$. *The algorithm* $\mathsf{Revoke}_{\mathsf{CA}}$ *allows to revoke a given public key* $\mathsf{pk}_{\mathsf{User}}$. *In a nutshell, the CA no longer agrees to start a signing protocol for revoked* $\mathsf{pk}_{\mathsf{User}}$. *Thus, revocation of a* $\mathsf{pk}_{\mathsf{User}}$ *does not affect already ongoing signing sessions for this* $\mathsf{pk}_{\mathsf{User}}$. *This algorithm outputs nothing:*

$$\perp \leftarrow \mathsf{Revoke}_{\mathsf{CA}}(\mathsf{pk}_{\mathsf{User}})$$

$\mathsf{Verify}_{\mathsf{A}}$. *The algorithm* $\mathsf{Verify}_{\mathsf{A}}$ *outputs a decision bit* $d \in \{\mathsf{false}, \mathsf{true}\}$, *indicating the validness of the signature* σ, *with respect to* $\mathsf{pk}_{\mathsf{CA}}$, pk_s, time, *and* m:

$$d \xleftarrow{\$} \mathsf{Verify}_{\mathsf{A}}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{User}}, m, \mathsf{time}, \sigma)$$

7.2.2 Security of CA-Assisted Signatures

We now define the formal requirements for CA-assisted signatures. In a nutshell, those are correctness, unforgeability against malicious users and CAs, and blindness/privacy against CAs and outsiders.

Correctness. As usual, correctness of any CASIG is required. In particular, it is required that for all security parameters $\lambda \in \mathbb{N}$ $\mathsf{Verify}_{\mathsf{A}}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{User}}, m, \mathsf{time}, \sigma) = \mathsf{true}$, for all $(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{U}}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CA}}(1^\lambda)$, for all $m \in \mathcal{MS}$, for all $\mathsf{time} \in \mathbb{N}$, and for all $(\perp, \sigma) \xleftarrow{\$} \langle \mathcal{CA}(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_s, \mathsf{time}), \mathcal{U}(\mathsf{sk}_s, \mathsf{pk}_{\mathsf{User}}, m, \mathsf{time}) \rangle$, while $\mathsf{pk}_{\mathsf{User}}$ was not revoked *before the signature generation request.* The probability space is here given by all random coins in all involved algorithms.

Unforgeability. Unforgeability of CA-assisted signatures covers two aspects. On the one hand, a malicious user must not be able to fake signatures of the CA. On the other hand, a malicious CA must not be able to impersonate a user. Together, those two definitions clearly also imply that an outsider is not able to forge any valid signatures.

For signer unforgeability, the adversary is allowed to obtain arbitrarily many signatures on arbitrary messages, and public keys, of its choice. Furthermore, for every signature, the adversary may define the current time (except that it may not turn back the time). Also, it can generate and revoke user keys at convenience. Similarly to Definition 7.2, the adversary now wins if it can output more message/signature pairs than it queried from the oracle; furthermore, each of those pairs must only verify for a public key and time that have been used in a signing query. Finally, signatures may only verify if the corresponding user public key has not been revoked before starting the respective signing session. For simplicity, it is defined that if a

Experiment $\mathsf{seUNF\text{-}CMA}^{\mathsf{CASIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{CA}}(1^\lambda)$
$\mathsf{time} \leftarrow 0$
$((m_1, \sigma_1, \mathsf{time}_1, \mathsf{pk}_1^*), \ldots, (m_\ell, \sigma_\ell, \mathsf{time}_\ell, \mathsf{pk}_\ell^*)) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\langle \mathcal{CA}(\mathsf{sk}_{\mathsf{CA}}, \cdot, \mathsf{time}), \cdot \rangle^\infty}, \mathcal{O}^{\mathsf{Timestamp}(\cdot)}, \mathcal{O}^{\mathsf{Revoke}_{\mathsf{CA}}(\cdot)}}(\mathsf{pk}_{\mathsf{CA}})$
where oracle $\mathsf{Timestamp}$ on input time':
  if $\mathsf{time}' \leq \mathsf{time}$, ignore
  let $\mathsf{time} \leftarrow \mathsf{time}'$
return 1, if $\forall i \in \{1, 2, \ldots, \ell\} : \mathsf{Verify}_{\mathsf{A}}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_i^*, m_i, \mathsf{time}_i, \sigma_i) = \mathsf{true}$
  and oracle $\langle \mathcal{CA}(\mathsf{sk}, \cdot, \cdot), \cdot \rangle$ finished less than ℓ times, and
  all $(m_i, \mathsf{time}_i, \mathsf{pk}_i)$ are pairwise distinct
return 1, if $\mathsf{Verify}_{\mathsf{A}}(\mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_1^*, m_1, \mathsf{time}_1, \sigma_1) = \mathsf{true}$,
  and pk_1^* was revoked before time_1
return 0

Figure 7.4: CASIG Signer Unforgeability

signing oracle is tagged as "non-called", if the corresponding public key was revoked before the current time. In the case that revocation and signing were done at the very same point in time, the resulting signature is not considered a forgery even if the revocation request was submitted first in the experiment; one the one hand, this is a purely academic issue anyways, and on the other hand "before" and "after" do not have any semantics within a fixed point in time. Of course, one could define that every call to same oracle increments the current time, as done for credential definitions [CKL+15]. However, it seems to be more natural to let the adversary decide.

Definition 7.6 (Signer Unforgeability)**.** *A CA-assisted signature scheme* CASIG *is signer unforgeable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr[\mathsf{seUNF\text{-}CMA}^{\mathsf{CASIG}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 7.4.

Complementary to signer unforgeability, it is also required that the CA cannot generate valid signatures for a specific user without its contribution. Therefore, the adversary (controlling the CA) can obtain arbitrarily many signatures for a user public key $\mathsf{pk}_{\mathsf{User}}$, where again $\mathcal{A}$ has full control over time. The adversary now wins if it can output a signature on a message that was not asked for that specific define point in time. This definition is similar to the standard definition of unforgeability, refer to Definition 2.17.

Note that as before, the adversary is allowed to interleave signing queries. Further note that the given definition is only presented in its weak formulation, i.e., the adversary is allowed to output fresh signatures for message/time pairs for which it obtained honest signatures. Extending the definition to strong unforgeability is straightforward.

Definition 7.7 (CA Unforgeability)**.** *A CA-assisted signature scheme* CASIG *is CA unforgeable,*

Experiment $\mathsf{ceUNF\text{-}CMA}^{\mathsf{CASIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{U}}(1^\lambda)$
$\mathsf{time} \leftarrow 0$
$(m^*, \sigma^*, \mathsf{time}^*, \mathsf{pk}^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\langle\cdot,\mathcal{U}(\mathsf{sk}_{\mathsf{User}},\cdot,\cdot,\mathsf{time})\rangle^\infty}, \mathcal{O}^{\mathsf{Timestamp}(\cdot)}}(\mathsf{pk}_{\mathsf{User}})$
where oracle $\mathsf{Timestamp}$ on input time':
  if $\mathsf{time}' \leq \mathsf{time}$, ignore
  let $\mathsf{time} \leftarrow \mathsf{time}'$
return 1, if $\mathsf{Verify}_{\mathsf{A}}(\mathsf{pk}^*, \mathsf{pk}_{\mathsf{User}}, m^*, \mathsf{time}^*, \sigma^*) = \mathsf{true}$,
  and oracle $\langle\cdot,\mathcal{U}(\mathsf{sk}_{\mathsf{User}},\cdot,\cdot,\cdot)\rangle$ was never queried for $(\mathsf{pk}^*, m^*, \mathsf{time}^*)$.
return 0

Figure 7.5: CASIG CA Unforgeability

if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:

$$\Pr[\mathsf{ceUNF\text{-}CMA}^{\mathsf{CASIG}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 7.5.

Blindness. Blindness is concerned with the privacy of the user towards the CA. While a secure CA-assisted signature scheme must satisfy both aspects of unforgeability, blindness comes in two flavors giving different privacy guarantees.

The first flavor, called *CA blindness*, is similar in spirit to Definition 7.3. There, the CA (controlled by the adversary) may trigger signing protocols on two messages of its choice in a random order, gets the resulting signatures, and then needs to link the transcripts to the messages.

In the second flavor, called *CA weak-blindness*, it is only required that the adversary does not learn which message it signed. In particular, the adversary does not gain access to the signatures, and may only trigger a single signing query. It is easy to see that CA blindness implies CA weak-blindness, but not vice versa. The decision which level of blindness/privacy is required must be made on a case-to-case basis, depending on the concrete use case.

Similar to Definition 7.3, the adversary is restricted to a single interaction with each oracle in the blindness definitions. However, blindness against multiple protocol runs directly follows from a simple hybrid argument.

Definition 7.8 (CA Blindness)**.** *A CA-assisted signature scheme* CASIG *is CA blind, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{CA\text{-}Blindness}^{\mathsf{CASIG}}_{\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 7.6.

Experiment $\mathsf{CA\text{-}Blindness}^{\mathsf{CASIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{U}}(1^\lambda)$
$(\mathsf{pk}^*, m_0, m_1, \mathsf{time}, state_1) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\mathsf{User}})$
$b \xleftarrow{\$} \{0,1\}$
$state_2 \xleftarrow{\$} \mathcal{A}^{\langle\cdot,\mathcal{U}_0(\mathsf{sk}_{\mathsf{User}},\mathsf{pk}^*,m_b,\mathsf{time})\rangle^1,\langle\cdot,\mathcal{U}_1(\mathsf{sk}_{\mathsf{User}},\mathsf{pk}^*,m_{1-b},\mathsf{time})\rangle^1,\mathsf{Revoke}_{\mathsf{CA}}(\cdot)}(state_1)$
let σ_0, and σ_1 denote the output of $\mathcal{U}_0$, and $\mathcal{U}_1$.
If $\sigma_0 = \bot \vee \sigma_1 = \bot$, let $\sigma \leftarrow \bot$.
Else, set $\sigma \leftarrow (\sigma_b, \sigma_{b-1})$
$a \xleftarrow{\$} \mathcal{A}(state_2, \sigma)$
return 1, if $a = b$
return 0

Figure 7.6: CASIG CA Blindness

Experiment $\mathsf{CA\text{-}WBlindness}^{\mathsf{CASIG}}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{U}}(1^\lambda)$
$(\mathsf{pk}^*, m_0, m_1, \mathsf{time}, state) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}_{\mathsf{User}})$
$b \xleftarrow{\$} \{0,1\}$
$a \xleftarrow{\$} \mathcal{A}^{\langle\cdot,\mathcal{U}(\mathsf{sk}_{\mathsf{User}},\mathsf{pk}^*,m_b,\mathsf{time})\rangle^1,\mathsf{Revoke}_{\mathsf{CA}}(\cdot)}(state)$
return 1, if $a = b$
return 0

Figure 7.7: CASIG Weak CA Blindness

Definition 7.9 (CA Weak-Blindness)**.** *A CA-assisted signature scheme* CASIG *is weakly CA-blind, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr[\mathsf{CA\text{-}WBlindness}^{\mathsf{CASIG}}_{\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure 7.7.

A CA-assisted signature scheme CASIG is *secure and (weakly) blind*, if it is correct, signer unforgeable, CA unforgeable, and CA (weakly) blind.

7.3 Constructions

It is now shown how to come up with constructions achieving what is defined. First, a generic construction is presented, which, depending on the used building blocks, achieves weaker, or stronger resp., privacy notions. The reductions are tight, i.e., there are no reduction losses, and a probability analysis in the proofs is omitted.

7.3.1 Generic Construction Idea

The generic idea of the construction is introduced first. Then, two different derivations of the generic constructions are given, which are instantiated with different building blocks. Both constructions offer the same unforgeability guarantees, but offer a different level of privacy.

In a nutshell, the scheme lets a CA contribute to signature generation, but only if the public key of the requester is not revoked at the time time of the signature request. The CA can then also add some additional information to the final signature such as certificates, and the like. However, from a privacy point of view, it is also required that the CA does not learn which messages m are signed, which reflects (weak-)blindness.

On the one hand, the signer commits to a message. The CA then signs this commitment (and the signer's public key), if, and only if, the given public key is not revoked. The user, on the other hand, creates an additional signature around the received signature from the CA to protect against bogus CAs. Clearly, there is no joint setup, and thus key generation can be done offline, which is not possible in current schemes. It is stressed that revoking a public key is simply sending the CA a message "My pk has been revoked", possibly containing a proof of knowledge, which is not necessarily zero-knowledge.

Note, the parties do not need to communicate using a secure channel.

7.3.2 The Constructions

Now, two constructions are presented. The first one is very efficient, while the second one is a bit more complex, but gives better privacy guarantees.

Construction 7.10 (Weakly-Blind Construction)**.** *Let* CASIG *such that:*

$\mathsf{KeyGen}_{\mathsf{U}}$**.** *Generate a key-pair of a digital signature scheme, i.e., return* $(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.

$\mathsf{KeyGen}_{\mathsf{CA}}$**.** *Generate a key-pair of a digital signature scheme* $(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$*, and the public parameters* $\mathsf{pp}_{\mathsf{CC}} \xleftarrow{\$} \mathsf{ParGen}_{\mathsf{CC}}(1^\lambda)$ *of a commitment scheme. Return* $(\mathsf{sk}_{\mathsf{CA}}, (\mathsf{pk}_{\mathsf{CA}}, \mathsf{pp}_{\mathsf{CC}}))$.

$\langle \mathcal{CA}, \mathcal{U} \rangle$**.** *See Figure 7.8.*

$\mathsf{Verify}_{\mathsf{A}}$**.** *To verify a signature* $\sigma = (\sigma', \sigma_c, C, O, \mathsf{time})$ *w.r.t.* m, $\mathsf{pk}_{\mathsf{CA}}$*, and* $\mathsf{pk}_{\mathsf{User}}$*, check that* $m = \mathsf{Open}_{\mathsf{CC}}(\mathsf{pp}_{\mathsf{CC}}, C, O)$*, and* $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{CA}}, (C, \mathsf{time}, \mathsf{pk}_{\mathsf{User}}), \sigma_c) = \mathsf{true}$*, and* $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{User}}, (\sigma_c, \mathsf{time}, m, C, O, \mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{User}}), \sigma') = \mathsf{true}$*. If all checks pass, output* true*, and* false *otherwise.*

Theorem 7.11. *If* DSIG *and* CC *are secure, then the given construction is secure and weakly blind.*

Proof. Correctness follows by inspection. Thus, only consider signer unforgeability, CA unforgeability, weak blindness need to be considered. Each property is proven on their own.

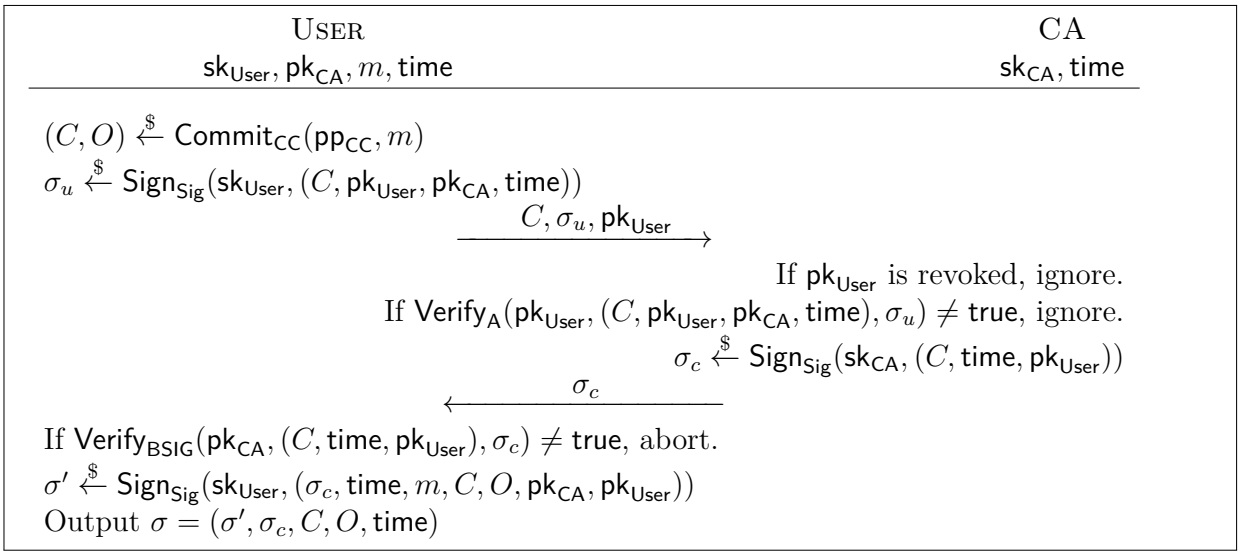

Figure 7.8: CA-Assisted signing with Weak Blindness

Signer Unforgeability. Let $\mathcal{A}$ be an adversary which can break the signer unforgeability of the construction. We can then construct an adversary $\mathcal{B}$ which either breaks the binding property of CC, or the unforgeability of the signature scheme DSIG used by the CA. Assume towards contradiction that there is a signature σ on the message $(\sigma_c, \mathsf{time}, m, C, O, \mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}^*)$, where σ_c is a signature on the message (C, time), but also a signature σ' for $(\sigma_c, \mathsf{time}', m', C, O, \mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}'^*)$, where $(m, \mathsf{time}, \mathsf{pk}^*) \neq (m', \mathsf{time}', \mathsf{pk}'^*)$. Hence, there are two different messages which "are in" the same commitment C. Clearly, this breaks the binding property of the commitment scheme used. In the second case, i.e., there is a new commitment C' for $(m, \mathsf{time}) \neq (m', \mathsf{time}')$ never signed by the CA, the adversary must have been able to forge a signature σ'_c. This also accounts for a revoked public key. In both cases building a reduction is trivial and therefore omitted.

CA Unforgeability. This case is trivial as well. If the adversary $\mathcal{A}$ can come up with a signature on a message $(\sigma_c, \mathsf{time}, m, C, O, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{User}})$, where $(m, \mathsf{time}, \mathsf{pk}^*)$ was never signed, then it can break the unforgeability of the used signature scheme. Again, a reduction is straightforward.

CA Weak-Blindness. Trivial, as CC is perfectly hiding, and therefore σ_u is independent of m, which is the only information sent to the CA, i.e., $\mathcal{A}$, which is the only value computed from m. □

Construction 7.12 (Blind Construction)**.** *Let* CASIG' *such that:*

$\mathsf{KeyGen}_{\mathsf{U}}$**.** *Generate a key-pair of a digital signature scheme, i.e., return* $(\mathsf{sk}_{\mathsf{User}}, \mathsf{pk}_{\mathsf{User}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{Sig}}(1^\lambda)$.

$\mathsf{KeyGen}_{\mathsf{CA}}$**.** *Generate a key-pair of a partially blind signature scheme, i.e., return* $(\mathsf{sk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{CA}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{BSIG}}(1^\lambda)$.

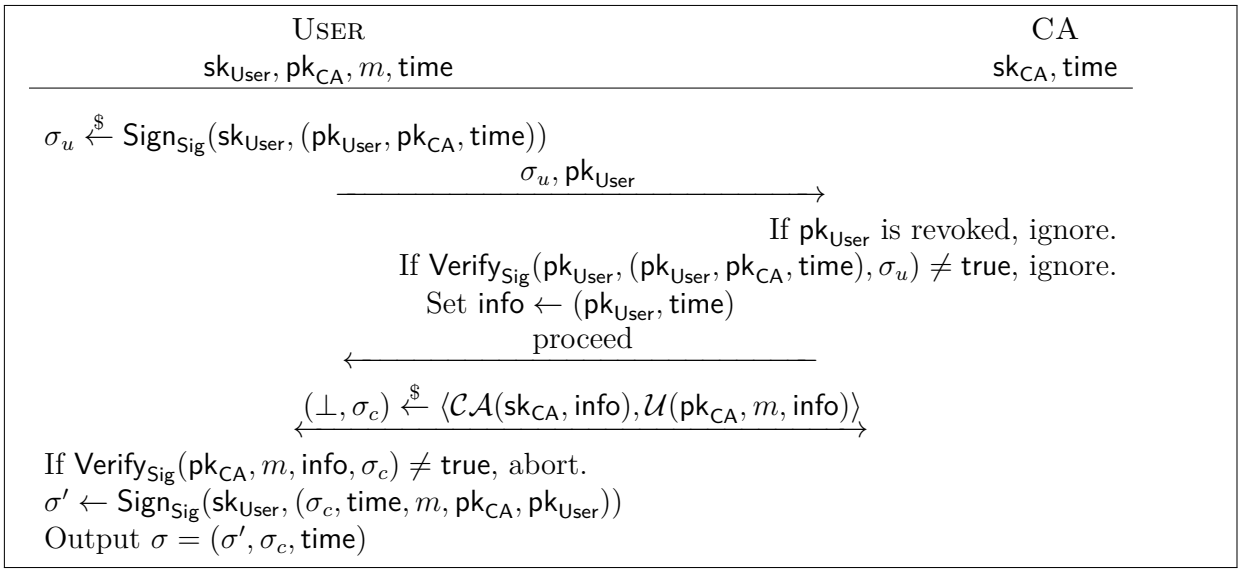

Figure 7.9: CA-Assisted signing with Blindness $\langle \mathcal{CA}, \mathcal{U} \rangle$. *See Figure 7.9.*

$\mathsf{Verify}_{\mathsf{A}}$. *To verify a signature* $\sigma = (\sigma', \sigma_c, \mathsf{time})$ *with respect to* m, $\mathsf{pk}_{\mathsf{CA}}$, *and* $\mathsf{pk}_{\mathsf{User}}$, *check that* $\mathsf{Verify}_{\mathsf{Sig}}(\mathsf{pk}_{\mathsf{User}}, (\sigma_c, \mathsf{time}, m, \mathsf{pk}_{\mathsf{CA}}, \mathsf{pk}_{\mathsf{User}}), \sigma') = \mathsf{true}$, *and* $\mathsf{Verify}_{\mathsf{BSIG}}(\mathsf{pk}_{\mathsf{CA}}, m, (\mathsf{pk}_{\mathsf{User}}, \mathsf{time}), \sigma_c) = \mathsf{true}$. *If all checks pass, output* true, *and* false *otherwise.*

Theorem 7.13. *If* DSIG *and* BSIG *are secure, then the construction is secure and blind.*

Proof. Again, correctness follows by inspection. It remains to prove CA unforgeability, signer unforgeability, and blindness.

Signer Unforgeability. Let $\mathcal{A}$ be an adversary which can break the signer unforgeability of the construction. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of the partially blind signature scheme. $\mathcal{B}$ receives pk from the BSIG to forge, and embeds the received pk into the public key $\mathsf{pk}_{\mathsf{CA}}$. It simply follows the protocol, and uses its own oracle to get signatures. If a given pk_i is revoked, $\mathcal{B}$ no longer accepts new signing sessions. Eventually, $\mathcal{A}$ outputs $((m_1, \sigma_1, \mathsf{info}_1, \mathsf{pk}_1), \ldots, (m_\ell, \sigma_\ell, \mathsf{time}_\ell, \mathsf{pk}_\ell))$. Clearly, if pk_1 was revoked, $\mathcal{B}$ never asked its own oracle to generate a signature for $(m_1, (\mathsf{pk}_1, \mathsf{time}_1))$, and can thus return all successful runs, and $(m_1, \sigma_1, \mathsf{time}_1, \mathsf{pk}_1)$, as for (m_1, time_1) is fresh by assumption, as $\mathcal{B}$ never queries its own oracle any longer for fresher time.

CA Unforgeability. Essentially the same reduction as for the weakly blind scheme.

CA Blindness. Let $\mathcal{A}$ be an adversary which breaks the CA blindness of the scheme. We can then construct an adversary which breaks the blindness of the used BSIG. $\mathcal{B}$ proceeds as follows. It generates $\mathsf{pk}_{\mathsf{User}}$ honestly, which it also gives to $\mathcal{A}$, receiving $(\mathsf{pk}^*, \{m_0, m_1\}, \mathsf{time}, \mathsf{state}_1)$. It then gives state_1 to $\mathcal{A}$, and interacts with its own oracles like $\mathcal{A}$ does with his using m_0 and m_1, but uses $(\mathsf{pk}_{\mathsf{User}}, \mathsf{time})$ as info. If $\mathcal{A}$ is finished it returns state_2, and $\mathcal{B}$ subsequently receives

(σ_1, σ_2) from its own challenger. Then, $\mathcal{B}$ gives $\mathcal{A}$ state_2, and (σ_1, σ_2) to $\mathcal{A}$. Whatever $\mathcal{A}$ then outputs, is also output by $\mathcal{B}$. □

7.3.2.1 Efficiency

In the first protocol message the user essentially proves knowledge of the secret key. If the signature on time is not valid, the protocol can directly be aborted. This prohibits that outsiders use the CA to check whether a given certificate is revoked. If this is not wanted for performance reasons, leaving this step out is also possible.

Clearly, both constructions require that a verifier needs to verify two signatures, while the CA has to generate a signature. However, considering that the CA has to vouch that a given certificate was not revoked, it has to generate a signature anyway, if the revocation information needs to be up-to-date, which clearly needs to be verified as well. In other words, the construction is already more efficient after the first signature verification. Moreover, compared to the approach by Boneh et al. [BDTW01], an outsider can trivially derive whether the given protocol was used to generate the signature, which in turn increases trust in the signature itself, as the verifier can also decide whether it accepts a given $\mathsf{pk}_{\mathsf{CA}}$ as trustworthy.

7.3.3 Extensions

Next is discussed informally how the basic constructions can be extended to account for additional use-cases.

Signer-Anonymity. Both constructions give message-privacy guarantees to the user, they reveal the identity of the signing party to the CA. If this poses a potential privacy problem, it can be mitigated as follows, for instance for the weakly-blind construction, refer to Figure 7.8. The commitment is extended to also commit to $\mathsf{pk}_{\mathsf{User}}$. Then, instead of signing the tuple in Figure 7.8 in the first step, one computes a signature proof of knowledge proving in zero-knowledge that one knows the secret key corresponding to the public key in the commitment, and that this public key is not on the blacklist. This can be done using similar techniques as used in Idemix [CL02a].

Revocation into the Past. Both constructions are well-suited for situations where signing keys should simply be deactivated, e.g., when an employee leaves a company. However, in certain situations, it is also necessary to revoke "into the past" in order to also invalidate signatures issued between key leakage and revocation, refer to Figure 7.1. In this case, the CA has to publish a list of revoked keys together with time-stamps of their revocation moment; upon verification, only signatures issued before this point in time would be accepted. From a complexity point of view this solution is similar to the combination of black-list based PKIs and time-stamping authorities, i.e., interaction is needed upon signing and verification.

Message Policies. One could also require that the signer proves (in zero-knowledge) that the message to be signed follows certain restrictions, e.g., that a company policy is followed. Only if the proof is valid, and the public key is not revoked, the server contributes to signature-generation. For example, a policy may be that a normal employee can only sign contracts below $1,000. This can even be done on a per-public-key basis. The size of signatures does not grow by

this extension, and also the verification costs do not increase. Furthermore, the policy trivially remains hidden from the verifier.

Further extending the scheme efficiently such that also the CA does not learn any information about the policy remains a challenging open problem.

Robustness. Even though the security model is fixed for one signer and one CA, one can of course switch to a different CA on-the-fly. This protects against offline CAs, as one can simply use another one. In particular, a user can use a single signing key with different CAs, who act as revocation authorities for different domains (e.g., across different companies). Revocation by one CA does not affect other CAs, if no synchronization for revocation between CAs is used. Security follows by a simple hybrid-argument.

Threshold Scheme. Related to the prior idea is an extension to threshold-cryptography. Namely, one could require that at least n-out-of-m servers need to participate in order to achieve robustness against offline or corrupted servers.

7.4 Conclusion

This chapter introduced the notion of CA-Assisted Signatures. These signatures enable the revocation of signing-rights if a secret is corrupted. This is achieved by letting a CA contribute to signature generation, vouching that the used public key was not revoked. Thus, signatures remain valid even after revocation of the certificate. Moreover, the CA can add timestamps, while neither the verifier nor the CA need to be online for verification. This has the additional benefit that verification requires less effort to check the validity of the signature. Furthermore, various extensions were proposed, increasing the privacy guarantees of our basic constructions.

The constructions do not pose any non-standard requirements to the signature scheme used by the user. In particular, existing signing infrastructures could thus easily be adapted to the design, while the users can continue using their favorite signature scheme. However, it remains an open question whether the idea can be extended for a chain of authorities or even generalized for other access structures.

Chapter 8

Concluding Remarks

This thesis proposed a number of protocols which solve some real-life challenges, intended for direct personal use. The proposed protocols address secure handling of passwords used for authentication and digital signature schemes with additional features, while being efficient and also directly deployable in today's infrastructure.

In particular, the first contributions of this thesis are two UC-secure distributed password-based single sign-on protocols, where the password check is distributed among multiple servers to avoid that servers can impersonate users or that servers can mount attacks on the password (attempts). However, it remains a challenging task to make the protocol secure against adaptive adversaries.

The second contribution are virtual smart-cards, where a user can use its personal device, such as a smart-phone, to sign arbitrary messages, but additionally needs to enter a password. The password is checked at an additional server which also contributes to signature generation, if the password entered was correct. The server does not learn the messages being signed, while neither entity alone has enough information to mount offline attacks on the password (attempts). Moreover, to account for the case of a lost device, the proposed protocol is secure against adaptive corruptions in a new corruption model, where not all prior input and output is given to the simulator. This is necessary to model that neither the used passwords nor the signed messages are not given to the adversary. However, it remains an open challenge to generalize the protocol to more than one server and to add unlinkability.

The third contribution is a new non-interactive UC-secure definition of non-committing encryption for adaptive adversaries which does not require secure erasures. Such a non-interactive definition is necessary, if the environment needs explicit access to ciphertexts or secure erasures are not a realistic assumption. This is demonstrated by providing the first definition of UC-secure signcryption secure against adaptive adversaries without secure erasures. As a sanity check, the new definition is proven to be equivalent to a game-based definition which is compared to a number of existing definitions addressing adaptive adversaries. It is also shown that this definition is only realizable in idealized models. However, the proposed definition does not account for key-dependent messages, which may allow for additional application scenarios.

The fourth contributions are chameleon-hashes with ephemeral trapdoors. These chameleon-hashes only allow to find collisions, if two trapdoors at the same time are known. One trapdoor is long-term, while the second one is generated at generation of the hash. Such a hash allows to prohibit that the holder of the long-term trapdoor can find collisions by keeping the ephemeral

trapdoor secret. The main application scenario is invisible sanitizable signature schemes. In these signature schemes, a semi-trusted third party, named the sanitizer, can alter signer-chosen blocks to arbitrary bit-strings, while also hiding which parts are actually admissible from outsiders. However, it is still an open problem how to construct sanitizable signature schemes which are invisible and unlinkable simultaneously.

The fifth contributions are efficient protocols for signing-right revocation. In these protocols, a revocation authority helps generating a signature and vouches that, at the time of signature generation, the public key was not revoked. This allows verifiers to skip checking the revocation status of a signature at signature verification, thus no longer requiring network access. Moreover, in the proposed protocols, the revocation authority neither learns the message signed nor requires to store any account information besides the revocation status, while the revocation authority can also add additional information, e.g., a time-stamp. An additional extension also prohibits that the revocation authority can link signatures with signature generation protocol transcripts.

From a more general point of view, none of the proposed protocols consider leakage, which is also not considered in UC. Moreover, it is also cumbersome to manually write and check proofs of protocols. Thus, having an *easy* to use theorem prover would be very beneficial.

Bibliography

[AB13] Prabhanjan Ananth and Raghav Bhaskar. Non Observability in the Random Oracle Model. In Willy Susilo and Reza Reyhanitabar, editors, *Provable Security - 7th International Conference, ProvSec 2013, Melaka, Malaysia, October 23-25, 2013. Proceedings*, volume 8209 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2013. (cited on page 27).

[ABC+15] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on Authenticated Data. *J. Cryptology*, 28(2):351–395, 2015. (cited on pages 7 and 200).

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In Vitaly Shmatikov, editor, *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering, FMSE 2008, Alexandria, VA, USA, October 27, 2008*, pages 1–10. ACM, 2008. (cited on page 35).

[ACDD14] Masayuki Abe, Jan Camenisch, Rafael Dowsley, and Maria Dubovitskaya. On the Impossibility of Structure-Preserving Deterministic Primitives. In Lindell [Lin14], pages 713–738. (cited on page D-1).

[ACdMT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable Signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2005. (cited on pages 7, 198, 199, 200, 203, 208 and 220).

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In Yung [Yun02], pages 417–432. (cited on page 104).

[AdM04] Giuseppe Ateniese and Breno de Medeiros. On the Key Exposure Problem in Chameleon Hashes. In Blundo and Cimato [BC05], pages 165–179. (cited on pages 168, 169 and 171).

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified Threshold RSA with Adaptive and Proactive Security. In Vaudenay [Vau06], pages 593–611. (cited on page 94).

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the Security of Joint Signature and Encryption. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002. (cited on pages 19, 103 and 161).

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-Preserving Signatures and Commitments to Group Elements. In Rabin [Rab10], pages 209–236. (cited on page 7).

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups. In Rogaway [Rog11], pages 649–666. (cited on page 42).

[AGK08] Masayuki Abe, Rosario Gennaro, and Kaoru Kurosawa. Tag-KEM/DEM: A New Framework for Hybrid Encryption. *J. Cryptology*, 21(1):97–130, 2008. (cited on page 208).

[ALP12] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on Authenticated Data: New Privacy Definitions and Constructions. In Wang and Sako [WS12], pages 367–385. (cited on page 7).

[AMVA17] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable Blockchain - or - Rewriting History in Bitcoin and Friends. In Sabelfeld and Smith [SS17], pages 111–126. (cited on pages 169, 170 and 179).

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably Secure Partially Blind Signatures. In Bellare [Bel00], pages 271–286. (cited on pages 220, 221 and 222).

[AP03] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless Public Key Cryptography. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer, 2003. (cited on page 220).

[App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 805–816. ACM, 2012. (cited on page 15).

[BB04] Kemal Bicakci and Nazife Baykal. Server Assisted Signatures Revisited. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 143–156. Springer, 2004. (cited on page 220).

[BBC+13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New Techniques for SPHFs and Efficient One-Round PAKE Protocols. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 449–475. Springer, 2013. (cited on page 35).

[BBD+10] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In Zhou and Yung [ZY10], pages 87–104. (cited on page 200).

[BBF13] Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of Black-Box Reductions, Revisited. In Sako and Sarkar [SS13], pages 296–315. (cited on page 10).

[BC05] Carlo Blundo and Stelvio Cimato, editors. *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, volume 3352 of *Lecture Notes in Computer Science*. Springer, 2005. (cited on pages 235 and 249).

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. (cited on page 168).

[BCD+17] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with Applications to Anonymity-Preserving Revocation. In Sabelfeld and Smith [SS17], pages 301–315. (cited on page 220).

[BCG07] Emmanuel Bresson, Dario Catalano, and Rosario Gennaro. Improved On-Line/Off-Line Threshold Signatures. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2007. (cited on page 168).

[BCJ+06] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of Kerberos 5. *Theor. Comput. Sci.*, 367(1-2):57–87, 2006. (cited on page 35).

[BCJ+11] Michael Backes, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. *Int. J. Inf. Sec.*, 10(2):107–134, 2011. (cited on page 35).

[BCK+14] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better Zero-Knowledge Proofs for Lattice Encryption and Their Application to Group Signatures. In Palash Sarkar and Tetsu Iwata, editors,

Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I, volume 8873 of *Lecture Notes in Computer Science*, pages 551–572. Springer, 2014. (cited on page 7).

[BCN+10] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get Shorty via Group Signatures without Encryption. In Garay and Prisco [GP10], pages 381–398. (cited on page 7).

[BDD+11] Feng Bao, Robert H. Deng, Xuhua Ding, Junzuo Lai, and Yunlei Zhao. Hierarchical Identity-Based Chameleon Hash and Its Applications. In Lopez and Tsudik [LT11], pages 201–219. (cited on page 168).

[BdM93] Josh Cohen Benaloh and Michael de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract). In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1993. (cited on page 220).

[BDPA14] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Making of KECCAK. *Cryptologia*, 38(1):26–60, 2014. (cited on page 27).

[BDT04] Dan Boneh, Xuhua Ding, and Gene Tsudik. Fine-grained control of security capabilities. *ACM Trans. Internet Techn.*, 4(1):60–82, 2004. (cited on page 220).

[BDTW01] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. A Method for Fast Revocation of Public Key Certificates and Security Capabilities. In Dan S. Wallach, editor, *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*. USENIX, 2001. (cited on pages 94, 217, 218, 219 and 231).

[BDWY12] Mihir Bellare, Rafael Dowsley, Brent Waters, and Scott Yilek. Standard Security Does Not Imply Security against Selective-Opening. In Pointcheval and Johansson [PJ12], pages 645–662. (cited on page 136).

[Bea97] Donald Beaver. Plug and Play Encryption. In Kaliski Jr. [Kal97], pages 75–89. (cited on page 136).

[Bea99] Paul Beame, editor. *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, IEEE Computer Society, 1999. (cited on page 253).

[Bei11] Amos Beimel. Secret-Sharing Schemes: A Survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology - Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, volume 6639 of *Lecture Notes in Computer Science*, pages 11–46. Springer, 2011. (cited on page 41).

[Bel00] Mihir Bellare, editor. *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*. Springer, 2000. (cited on pages 236, 250 and 253).

[BF01] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In Kilian [Kil01], pages 213–229. (cited on page 28).

[BF05] Alexandra Boldyreva and Marc Fischlin. Analysis of Random Oracle Instantiation Scenarios for OAEP and Other Practical Schemes. In Shoup [Sho05], pages 412–429. (cited on page 135).

[BF11a] Paul Baecher and Marc Fischlin. Random Oracle Reducibility. In Rogaway [Rog11], pages 21–38. (cited on page 27).

[BF11b] Dan Boneh and David Mandell Freeman. Homomorphic Signatures for Polynomial Functions. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer, 2011. (cited on page 7).

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of Sanitizable Signatures Revisited. In Jarecki and Tsudik [JT09], pages 317–336. (cited on pages 7, 169, 170, 172, 188, 199, 200, 202, 203, 208, 220 and I-1).

[BFLS09] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Santizable Signatures: How to Partially Delegate Control for Authenticated Data. In Arslan Brömme, Christoph Busch, and Detlef Hühnlein, editors, *BIOSIG 2009 - Proceedings of the Special Interest Group on Biometrics and Electronic Signatures, 17.-18. September 2009 in Darmstadt, Germany*, volume 155 of *LNI*, pages 117–128. GI, 2009. (cited on page 200).

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of Sanitizable Signatures. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2010. (cited on pages 198, 199, 204 and 206).

[BFM14] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Indistinguishability Obfuscation and UCEs: The Case of Computationally Unpredictable Sources. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2014. (cited on page 28).

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically Uncloneable Functions in the Universal Composition Framework. In Rogaway [Rog11], pages 51–70. (cited on pages 6 and 26).

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (Im)possibility of Obfuscating Programs. In Kilian [Kil01], pages 1–18. (cited on page 92).

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012. (cited on page 92).

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional Signatures and Pseudorandom Functions. In Hugo Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519. Springer, 2014. (cited on page 7).

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In Ning et al. [NSJ08], pages 417–426. (cited on page 220).

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003. (cited on pages 59 and 220).

[BH92] Donald Beaver and Stuart Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1992. (cited on pages 96 and 136).

[BH04] Michael Backes and Dennis Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. In Zhang and Zheng [ZZ04], pages 61–72. (cited on page 29).

[BHK12] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. On Definitions of Selective Opening Security. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 522–539. Springer, 2012. (cited on pages 134 and 136).

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating Random Oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2013. (cited on page 28).

[BHK15] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the Definition of IND-CCA: When and How Should Challenge Decryption Be Disallowed? *J. Cryptology*, 28(1):29–48, 2015. (cited on page 204).

[BHPS16] Arne Bilzhause, Manuel Huber, Henrich C. Pöhls, and Kai Samelin. Cryptographically Enforced Four-Eyes Principle. In *11th International Conference on Availability, Reliability and Security, ARES 2016, Salzburg, Austria, August 31 - September 2, 2016*, pages 760–767. IEEE Computer Society, 2016. (cited on page 200).

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the Impossibility of Tight Cryptographic Reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 273–304. Springer, 2016. (cited on page 9).

[BJR+06] John G. Brainard, Ari Juels, Ronald L. Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In Juels et al. [JWdV06], pages 168–178. (cited on pages 5 and 6).

[BJSL11] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 433–444. ACM, 2011. (cited on pages 35, 49 and 95).

[BJST08] Bruno Blanchet, Aaron D. Jaggard, Andre Scedrov, and Joe-Kai Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In Masayuki Abe and Virgil D. Gligor, editors, *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, Tokyo, Japan, March 18-20, 2008*, pages 87–99. ACM, 2008. (cited on page 35).

[BK11] Alexandra Boldyreva and Virendra Kumar. Provable-security analysis of authenticated encryption in Kerberos. *IET Information Security*, 5(4):207–219, 2011. (cited on page 35).

[BKPS16] Michael Till Beck, Stephan Krenn, Franz-Stefan Preiss, and Kai Samelin. Practical Signing-Right Revocation. In Franz and Papadimitratos [FP16], pages 21–39. (cited on page 217).

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003. (cited on page 8).

[BLR04] Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol Initialization for the Framework of Universal Composability. *IACR Cryptology ePrint Archive*, 2004:6, 2004. (cited on page 44).

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001. (cited on page D-1).

[Blu81] Manuel Blum. Coin Flipping by Telephone. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81, CRYPTO 81, IEEE Workshop on Communications Security, Santa Barbara, California, USA, August 24-26, 1981.*, pages 11–15. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981. (cited on page 23).

[BM91] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos Authentication System. In *Proceedings of the Usenix Winter 1991 Conference, Dallas, TX, USA, January 1991*, pages 253–268. USENIX Association, 1991. (cited on page 35).

[BM14] Christina Brzuska and Arno Mittelbach. Using Indistinguishability Obfuscation via UCEs. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 2014. (cited on page 92).

[BMS16] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable Functional Signatures. In Cheng et al. [CCPY16], pages 357–386. (cited on page 7).

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In Juels et al. [JWdV06], pages 390–399. (cited on page 220).

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *J. Cryptology*, 16(3):185–215, 2003. (cited on pages 14, 94, 106 and 220).

[Bol03] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In Yvo Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8,*

2003, Proceedings, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003. (cited on pages 106, 219 and 220).

[Bon03] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003. (cited on pages 248, 250 and 251).

[BOV14] Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors. *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*. Springer, 2014. (cited on pages 262 and 267).

[Boy89] Colin Boyd. Digital multisignatures. In H. Backer and F. Piper, editors, *Cryptography and Coding*, pages 241–246, 1989. (cited on pages 7 and 94).

[Boy08] Xavier Boyen. The Uber-Assumption Family. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, volume 5209 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008. (cited on page 8).

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In Preneel [Pre00], pages 139–155. (cited on page 35).

[BPS12] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Non-interactive Public Accountability for Sanitizable Signatures. In Sabrina De Capitani di Vimercati and Chris Mitchell, editors, *Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, volume 7868 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 2012. (cited on pages 198, 199, 200 and 206).

[BPS13] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In Sokratis K. Katsikas and Isaac Agudo, editors, *Public Key Infrastructures, Services and Applications - 10th European Workshop, EuroPKI 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*, volume 8341 of *Lecture Notes in Computer Science*, pages 12–30. Springer, 2013. (cited on pages 198, 199, 200, 202, 206 and J-1).

[BPS17] Arne Bilzhause, Henrich C. Pöhls, and Kai Samelin. Position Paper: The Past, Present, and Future of Sanitizable and Redactable Signatures. In *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017*, pages 87:1–87:9. ACM, 2017. (cited on page 200).

[BPW12a] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In Wang and Sako [WS12], pages 626–643. (cited on pages 40 and 83).

[BPW12b] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. *J. Cryptology*, 25(1):57–115, 2012. (cited on page 7).

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993. (cited on pages 27, 28, 83, 98, 129, 134, 135, 139, 140, G-1 and H-1).

[BR94] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994. (cited on page 135).

[BR96] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer, 1996. (cited on page 20).

[BR06] Mihir Bellare and Phillip Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In Vaudenay [Vau06], pages 409–426. (cited on page 9).

[Bra90] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990. (cited on pages 252 and 268).

[Brz13] Christina Brzuska. *On the Foundations of Key Exchange*. PhD thesis, TU Darmstadt, August 2013. (cited on page 10).

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic number theory, volume 1: efficient algorithms*. MIT Press, Cambridge, Massachusetts, 1996. (cited on page 173).

[BS01a] Mihir Bellare and Ravi S. Sandhu. The Security of Practical Two-Party RSA Signature Schemes. *IACR Cryptology ePrint Archive*, 2001:60, 2001. (cited on pages 95, 110, 113 and 114).

[BS01b] Emmanuel Bresson and Jacques Stern. Efficient Revocation in Group Signatures. In Kwangjo Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 190–206. Springer, 2001. (cited on page 8).

[BS05] Boaz Barak and Amit Sahai. How To Play Almost Any Mental Game Over The Net - Concurrent Composition via Super-Polynomial Simulation. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 543–552. IEEE Computer Society, 2005. (cited on page 10).

[BW13] David Bernhard and Bogdan Warinschi. Cryptographic Voting - A Gentle Introduction. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 167–211. Springer, 2013. (cited on page 10).

[Can00] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptology*, 13(1):143–202, 2000. (cited on page 9).

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001. (cited on pages 9, 24, 26, 32, 34, 93, 99, 103, 105, 133, 134, 149 and 152).

[Can04] Ran Canetti. Universally Composable Signature, Certification, and Authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, pages 219–233. IEEE Computer Society, 2004. (cited on pages 29, 103, 107 and 155).

[CBN06] Sherman S. M. Chow, Colin Boyd, and Juan Manuel González Nieto. Security-Mediated Certificateless Cryptography. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 508–524. Springer, 2006. (cited on page 220).

[CC08] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In Ning et al. [NSJ08], pages 109–118. (cited on page 35).

[CCGS10] Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential Authenticated Identification and Key Exchange. In Rabin [Rab10], pages 255–276. (cited on pages 26 and 35).

[CCPY16] Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors. *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, volume 9614 of *Lecture Notes in Computer Science*. Springer, 2016. (cited on pages 242 and 254).

[CDE+14] Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. *J. Inf. Sec. Appl.*, 19(1):25–44, 2014. (cited on pages 217 and 220).

[CDK+17] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-Hashes with Ephemeral Trapdoors - And Applications to Invisible Sanitizable Signatures. In Serge Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 152–182. Springer, 2017. (cited on page 167).

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved Non-committing Encryption with Applications to Adaptively Secure Protocols. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2009. (cited on pages 134 and 136).

[CDR16] Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. UC Commitments for Modular Protocol Design and Applications to Revocation and Attribute Tokens. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 208–239. Springer, 2016. (cited on page 8).

[CEK+16] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal Composition with Responsive Environments. In Cheon and Takagi [CT16], pages 807–840. (cited on pages 27 and 149).

[CEM16] Jan Camenisch, Robert R. Enderlein, and Ueli Maurer. Memory Erasability Amplification. In Zikas and Prisco [ZP16], pages 104–125. (cited on page 26).

[CEN15] Jan Camenisch, Robert R. Enderlein, and Gregory Neven. Two-Server Password-Authenticated Secret Sharing UC-Secure Against Transient Corruptions. In Katz [Kat15], pages 283–307. (cited on pages 27, 35, 94, 95 and 136).

[CES13] Jan Camenisch, Robert R. Enderlein, and Victor Shoup. Practical and Employable Protocols for UC-Secure Circuit Evaluation over $\mathbb{Z}_n$. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 19–37. Springer, 2013. (cited on page 95).

[CF01] Ran Canetti and Marc Fischlin. Universally Composable Commitments. In Kilian [Kil01], pages 19–40. (cited on page 32).

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively Secure Multi-Party Computation. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996. (cited on page 136).

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. (cited on pages 28 and 135).

[CH02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 21–30. ACM, 2002. (cited on pages 1, 7 and 220).

[CH06] Ran Canetti and Jonathan Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In Halevi and Rabin [HR06], pages 380–403. (cited on page 149).

[Cha82] David Chaum. Blind Signatures for Untraceable Payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.*, pages 199–203. Plenum Press, New York, 1982. (cited on pages 220 and 221).

[CHK05a] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-Secure, Non-interactive Public-Key Encryption. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2005. (cited on pages 134, 135, 136, 137, 139, 145, 147 and 149).

[CHK+05b] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally Composable Password-Based Key Exchange. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, 2005. (cited on pages 6, 35 and 93).

[CHMV17] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. A Universally Composable Treatment of Network Time. In Köpf and Chong [KC17], pages 360–375. (cited on page 43).

[CHYC05] Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. Two Improved Partially Blind Signature Schemes from Bilinear Pairings. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy, 10th Australasian Conference, ACISP 2005, Brisbane, Australia, July 4-6, 2005, Proceedings*, volume 3574 of *Lecture Notes in Computer Science*, pages 316–328. Springer, 2005. (cited on page 220).

[CJ10] Sébastien Canard and Amandine Jambert. On Extended Sanitizable Signature Schemes. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010. (cited on page 200).

[CJL12] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable Signatures with Several Signers and Sanitizers. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2012. (cited on page 200).

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a Global Random Oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608. ACM, 2014. (cited on pages 135 and 151).

[CJT02] Hung-Yu Chien, Jinn-ke Jan, and Yuh-Min Tseng. An Efficient and Practical Solution to Remote Authentication: Smart Card. *Computers & Security*, 21(4):372–375, 2002. (cited on page 6).

[CKL+15] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal Treatment of Privacy-Enhancing Credential Systems. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015. (cited on pages 8, 225, C-1, C-2 and D-1).

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing Chosen-Ciphertext Security. In Boneh [Bon03], pages 565–582. (cited on page 149).

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In Jarecki and Tsudik [JT09], pages 481–500. (cited on page 8).

[CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving Revocation with Efficient Update of Anonymous Credentials. In Garay and Prisco [GP10], pages 454–471. (cited on page 8).

[CKS11] Jan Camenisch, Stephan Krenn, and Victor Shoup. A Framework for Practical Universally Composable Zero-Knowledge Protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 449–467. Springer, 2011. (cited on pages 26 and 40).

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient Blind Signatures Without Random Oracles. In Blundo and Cimato [BC05], pages 134–148. (cited on page 220).

[CKY09] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the Portability of Generalized Schnorr Proofs. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 425–442. Springer, 2009. (cited on pages 82 and 83).

[CL02a] Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In Yung [Yun02], pages 61–76. (cited on pages 220 and 231).

[CL02b] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002. (cited on pages 7 and 42).

[CL05] Jan Camenisch and Anna Lysyanskaya. A Formal Treatment of Onion Routing. In Shoup [Sho05], pages 169–187. (cited on page 88).

[CL06] Melissa Chase and Anna Lysyanskaya. On Signatures of Knowledge. In Dwork [Dwo06], pages 78–96. (cited on page 40).

[CL12] Chin-Chen Chang and Chia-Yin Lee. A Secure Single Sign-On Mechanism for Distributed Computer Networks. *IEEE Trans. Industrial Electronics*, 59(1):629–637, 2012. (cited on pages 34 and 35).

[CL13] Sébastien Canard and Roch Lescuyer. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China - May 08 - 10, 2013*, pages 381–392. ACM, 2013. (cited on page 198).

[CLLN14] Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to Reconstruct Your Secrets from a Single Password in a Hostile Environment. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 256–275. Springer, 2014. (cited on pages 35, 36, 40, 49, 95, A-1 and A-2).

[CLM08] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor Sanitizable Signatures and Their Application to Content Protection. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York,*

NY, USA, June 3-6, 2008. Proceedings, volume 5037 of *Lecture Notes in Computer Science*, pages 258–276, 2008. (cited on page 200).

[CLMP13] Kai-Min Chung, Huijia Lin, Mohammad Mahmoody, and Rafael Pass. On the power of nonuniformity in proofs of security. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 389–400. ACM, 2013. (cited on page 25).

[CLN12] Jan Camenisch, Anna Lysyanskaya, and Gregory Neven. Practical yet universally composable two-server password-authenticated secret sharing. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 525–536. ACM, 2012. (cited on pages 35 and 95).

[CLN15] Jan Camenisch, Anja Lehmann, and Gregory Neven. Optimal Distributed Password Verification. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 182–194. ACM, 2015. (cited on pages 27 and 35).

[CLNR14] Jan Camenisch, Anja Lehmann, Gregory Neven, and Alfredo Rial. Privacy-Preserving Auditing for Attribute-Based Credentials. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2014. (cited on page 88).

[CLNS16] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. Virtual Smart Cards: How to Sign with a Password and a Server. In Zikas and Prisco [ZP16], pages 353–371. (cited on pages 35, 91 and 135).

[CLNS17] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. UC-Secure Non-Interactive Public-Key Encryption. In Köpf and Chong [KC17], pages 217–233. (cited on page 133).

[Cor00] Jean-Sébastien Coron. On the Exact Security of Full Domain Hash. In Bellare [Bel00], pages 229–235. (cited on pages 20 and 28).

[CR03] Ran Canetti and Tal Rabin. Universal Composition with Joint State. In Boneh [Bon03], pages 265–281. (cited on page 155).

[CRFG08] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Rosario Gennaro. Off-Line/On-Line Signatures: Theoretical Aspects and Experimental Results. In Ronald Cramer, editor, *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 101–120. Springer, 2008. (cited on page 168).

[CS97] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In Kaliski Jr. [Kal97], pages 410–424. (cited on pages 38, 42 and 82).

[CS03] Jan Camenisch and Victor Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In Boneh [Bon03], pages 126–144. (cited on pages 22, 85 and D-1).

[CSF+08] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russ Housley, and William Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818. (cited on page 217).

[CT16] Jung Hee Cheon and Tsuyoshi Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, 2016. (cited on pages 246 and 259).

[CTZD10] Xiaofeng Chen, Haibo Tian, Fangguo Zhang, and Yong Ding. Comments and Improvements on Key-Exposure Free Chameleon Hashing Based on Factoring. In Lai et al. [LYL11], pages 415–426. (cited on page 169).

[CvH91] David Chaum and Eugène van Heyst. Group Signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991. (cited on page 7).

[CZK04] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon Hashing Without Key Exposure. In Zhang and Zheng [ZZ04], pages 87–98. (cited on page 169).

[CZLC05] Tierui Chen, Bin B. Zhu, Shipeng Li, and Xueqi Cheng. ThresPassport - A Distributed Single Sign-On Service. In De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang, editors, *Advances in Intelligent Computing, International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part II*, volume 3645 of *Lecture Notes in Computer Science*, pages 771–780. Springer, 2005. (cited on pages 34 and 35).

[CZS+10] Xiaofeng Chen, Fangguo Zhang, Willy Susilo, Haibo Tian, Jin Li, and Kwangjo Kim. Identity-Based Chameleon Hash Scheme without Key Exposure. In Ron Steinfeld and Philip Hawkes, editors, *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2010. (cited on page 169).

[Dag13] Özgür Dagdelen. *The cryptographic security of the German electronic identity card.* PhD thesis, TU Darmstadt, July 2013. (cited on page 1).

[DDH+15] Denise Demirel, David Derler, Christian Hanser, Henrich C. Pöhls, Daniel Slamanig, and Giulia Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, EU Deliverable, 2015. (cited on page 200).

[DF89] Yvo Desmedt and Yair Frankel. Threshold Cryptosystems. In Brassard [Bra90], pages 307–315. (cited on pages 58, 94, 219, 220 and A-1).

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976. (cited on pages 7 and 8).

[DHPS15] David Derler, Christian Hanser, Henrich C. Pöhls, and Daniel Slamanig. Towards Authenticity and Privacy Preserving Accountable Workflows. In David Aspinall, Jan Camenisch, Marit Hansen, Simone Fischer-Hübner, and Charles D. Raab, editors, *Privacy and Identity Management. Time for a Revolution? - 10th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Edinburgh, UK, August 16-21, 2015, Revised Selected Papers*, volume 476 of *IFIP Advances in Information and Communication Technology*, pages 170–186. Springer, 2015. (cited on page 199).

[DHS14] David Derler, Christian Hanser, and Daniel Slamanig. Blank Digital Signatures: Optimization and Practical Experiences. In Jan Camenisch, Simone Fischer-Hübner, and Marit Hansen, editors, *Privacy and Identity Management for the Future Internet in the Age of Globalisation - 9th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Patras, Greece, September 7-12, 2014, Revised Selected Papers*, volume 457 of *IFIP Advances in Information and Communication Technology*, pages 201–215. Springer, 2014. (cited on page 198).

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2015. (cited on page 220).

[dKGHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A Practical Attack on the MIFARE Classic. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications, 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2008. (cited on page 6).

[DM09] Ivan Damgård and Gert Læssøe Mikkelsen. On the Theory and Practice of Personal Digital Signatures. In Jarecki and Tsudik [JT09], pages 277–296. (cited on page 95).

[dMPPS13] Hermann de Meer, Henrich C. Pöhls, Joachim Posegga, and Kai Samelin. Scope of Security Properties of Sanitizable Signatures Revisited. In *2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg,*

Germany, September 2-6, 2013, pages 188–197. IEEE Computer Society, 2013. (cited on pages 200 and 208).

[dMPPS14] Hermann de Meer, Henrich C. Pöhls, Joachim Posegga, and Kai Samelin. On the Relation between Redactable and Sanitizable Signature Schemes. In Jan Jürjens, Frank Piessens, and Nataliia Bielova, editors, *Engineering Secure Software and Systems - 6th International Symposium, ESSoS 2014, Munich, Germany, February 26-28, 2014, Proceedings*, volume 8364 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2014. (cited on page 200).

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved Non-committing Encryption Schemes Based on a General Complexity Assumption. In Bellare [Bel00], pages 432–450. (cited on page 136).

[DNRS99] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic Functions. In Beame [Bea99], pages 523–534. (cited on page 134).

[DS15] David Derler and Daniel Slamanig. Rethinking Privacy for Extended Sanitizable Signatures and a Black-Box Construction of Strongly Private Schemes. In Man Ho Au and Atsuko Miyaji, editors, *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *Lecture Notes in Computer Science*, pages 455–474. Springer, 2015. (cited on pages 200 and 204).

[DT03] Xuhua Ding and Gene Tsudik. Simple Identity-Based Cryptography with Mediated RSA. In Joye [Joy03], pages 193–210. (cited on page 220).

[Dwo06] Cynthia Dwork, editor. *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*. Springer, 2006. (cited on pages 249 and 254).

[End16] Robert R. Enderlein. *Practical composable cryptographic protocols resistant against adaptive attacks*. PhD thesis, ETH Zürich, 2016. (cited on pages 1 and 24).

[FF11] Marc Fischlin and Roger Fischlin. Efficient Non-Malleable Commitment Schemes. *J. Cryptology*, 24(1):203–244, 2011. (cited on page 24).

[FF15] Victoria Fehr and Marc Fischlin. Sanitizable Signcryption: Sanitization over Encrypted Data (Full Version). *IACR Cryptology ePrint Archive*, 2015:765, 2015. (cited on page 200).

[FH07] Dinei A. F. Florêncio and Cormac Herley. A large-scale study of web password habits. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 657–666. ACM, 2007. (cited on page 6).

[FHKP16] Georg Fuchsbauer, Felix Heuer, Eike Kiltz, and Krzysztof Pietrzak. Standard Security Does Imply Security Against Selective Opening for Markov Distributions. In Kushilevitz and Malkin [KM16], pages 282–305. (cited on pages 134 and 136).

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical Round-Optimal Blind Signatures in the Standard Model from Weaker Assumptions. In Zikas and Prisco [ZP16], pages 391–408. (cited on page 223).

[FHKW10] Serge Fehr, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Encryption Schemes Secure against Chosen-Ciphertext Selective Opening Attacks. In Gilbert [Gil10], pages 381–402. (cited on pages 96, 97, 134 and 136).

[Fis01] Marc Fischlin. *Trapdoor Commitment Schemes and Their Applications*. PhD thesis, University of Frankfurt, 2001. (cited on page 168).

[Fis06] Marc Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. In Dwork [Dwo06], pages 60–77. (cited on page 103).

[FKM+16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys. In Cheng et al. [CCPY16], pages 301–330. (cited on pages 199 and 206).

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the Non-malleability of the Fiat-Shamir Transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2012. (cited on page 83).

[FLR+10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random Oracles with(out) Programmability. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320. Springer, 2010. (cited on page 27).

[FO11] Marc Fischlin and Cristina Onete. Relaxed Security Notions for Signatures of Knowledge. In Lopez and Tsudik [LT11], pages 309–326. (cited on page 40).

[FP16] Michael Franz and Panos Papadimitratos, editors. *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings*, volume 9824 of *Lecture Notes in Computer Science*. Springer, 2016. (cited on pages 241 and 270).

[FPP+16] Christoph Frädrich, Henrich C. Pöhls, Wolfgang Popp, Noëlle Rakotondravony, and Kai Samelin. Integrity and Authenticity Protection with Selective Disclosure Control in the Cloud & IoT. In Kwok-Yan Lam, Chi-Hung Chi, and Sihan Qing,

editors, *Information and Communications Security - 18th International Conference, ICICS 2016, Singapore, November 29 - December 2, 2016, Proceedings*, volume 9977 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2016. (cited on page G-1).

[FRH+12] Muhammad Moazam Fraz, Paolo Remagnino, Andreas Hoppe, Bunyarit Uyyanonvara, Alicja R. Rudnicka, Christopher G. Owen, and Sarah A. Barman. Blood vessel segmentation methodologies in retinal images - A survey. *Computer Methods and Programs in Biomedicine*, 108(1):407–433, 2012. (cited on page 6).

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. (cited on pages 28 and 83).

[FS09] Marc Fischlin and Dominique Schröder. Security of Blind Signatures under Aborts. In Jarecki and Tsudik [JT09], pages 297–316. (cited on page 220).

[FS10] Marc Fischlin and Dominique Schröder. On the Impossibility of Three-Move Blind Signature Schemes. In Gilbert [Gil10], pages 197–215. (cited on page 220).

[Gam84] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984. (cited on pages 7, 24 and D-1).

[Gan95] Ravi Ganesan. Yaksha: augmenting Kerberos with public key cryptography. In James T. Ellis, David M. Balenson, and Robert W. Shirey, editors, *1995 Symposium on Network and Distributed System Security, (S)NDSS '95, San Diego, California, February 16-17, 1995*, pages 132–143. IEEE Computer Society, 1995. (cited on page 95).

[Gar95] Simson L. Garfinkel. *PGP - pretty good privacy: encryption for everyone (2. ed.).* O'Reilly, 1995. (cited on page 1).

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013. (cited on page 92).

[GGOT16] Esha Ghosh, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Verifiable Zero-Knowledge Order Queries and Updates for Fully Dynamic Lists and Trees. In Zikas and Prisco [ZP16], pages 216–236. (cited on page 200).

[Gil10] Henri Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010. (cited on pages 254 and 255).

[Gjø13] Kristian Gjøsteen. Partially blind password-based signatures using elliptic curves. *IACR Cryptology ePrint Archive*, 2013:472, 2013. (cited on page 95).

[GK96] Oded Goldreich and Hugo Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, 1996. (cited on page 24).

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 102–113. IEEE Computer Society, 2003. (cited on page 135).

[GK07] Kristian Gjøsteen and Lillian Kråkmo. Universally Composable Signcryption. In Javier Lopez, Pierangela Samarati, and Josep L. Ferrer, editors, *Public Key Infrastructure, 4th European PKI Workshop: Theory and Practice, EuroPKI 2007, Palma de Mallorca, Spain, June 28-30, 2007, Proceedings*, volume 4582 of *Lecture Notes in Computer Science*, pages 346–353. Springer, 2007. (cited on pages 155, 161 and 162).

[GLW09] Wei Gao, Fei Li, and Xueli Wang. Chameleon hash without key exposure based on Schnorr signature. *Computer Standards & Interfaces*, 31(2):282–285, 2009. (cited on page 169).

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. (cited on pages 17 and 21).

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985. (cited on page 24).

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. (cited on pages 7, 18, 198 and 217).

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987. (cited on pages 9 and 25).

[GNP+15] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration.

In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. (cited on page 7).

[Gos12] Jeremi Gosney. Password Cracking HPC. Passwords^12 Conference, 2012. (cited on pages 6, 34 and 92).

[GOT15] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Zero-Knowledge Authenticated Order Queries and Order Statistics on a List. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 149–171. Springer, 2015. (cited on page 200).

[GP10] Juan A. Garay and Roberto De Prisco, editors. *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*. Springer, 2010. (cited on pages 238 and 248).

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008. (cited on page 14).

[GQZ10] Junqing Gong, Haifeng Qian, and Yuan Zhou. Fully-Secure and Practical Sanitizable Signatures. In Lai et al. [LYL11], pages 300–317. (cited on pages 200, 202 and 208).

[GR13] Oded Goldreich and Ron D. Rothblum. Enhancements of Trapdoor Permutations. *J. Cryptology*, 26(3):484–512, 2013. (cited on page 17).

[GRJK00] Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and Efficient Sharing of RSA Functions. *J. Cryptology*, 13(2):273–300, 2000. (cited on page 94).

[Gro03] Thomas Groß. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. In *19th Annual Computer Security Applications Conference (ACSAC 2003), 8-12 December 2003, Las Vegas, NV, USA*, pages 298–307. IEEE Computer Society, 2003. (cited on page 35).

[Gro06] Jens Groth. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2006. (cited on pages 39 and 40).

[Gro15] Jens Groth. Efficient Fully Structure-Preserving Signatures for Large Messages. In Iwata and Cheon [IC15], pages 239–259. (cited on pages 42 and B-1).

[GT11] Kristian Gjøsteen and Øystein Thuen. Password-Based Signatures. In Svetla Petkova-Nikova, Andreas Pashalidis, and Günther Pernul, editors, *Public Key Infrastructures, Services and Applications - 8th European Workshop, EuroPKI 2011, Leuven, Belgium, September 15-16, 2011, Revised Selected Papers*, volume 7163 of *Lecture Notes in Computer Science*, pages 17–33. Springer, 2011. (cited on page 95).

[Gut02] Peter Gutmann. PKI: It's Not Dead, Just Resting. *IEEE Computer*, 35(8):41–49, 2002. (cited on page 218).

[GWX07] Wei Gao, Xueli Wang, and Dongqing Xie. Chameleon Hashes Without Key Exposure Based on Factoring. *J. Comput. Sci. Technol.*, 22(1):109–113, 2007. (cited on page 169).

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat Non-committing Encryption and Efficient Adaptively Secure Oblivious Transfer. In Halevi [Hal09], pages 505–523. (cited on page 136).

[Hal09] Shai Halevi, editor. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*. Springer, 2009. (cited on page 258).

[HJR16] Dennis Hofheinz, Tibor Jager, and Andy Rupp. Public-Key Encryption with Simulation-Based Selective-Opening Security and Compact Ciphertexts. In Hirt and Smith [HS16], pages 146–168. (cited on page 136).

[HKY15] Lucjan Hanzlik, Miroslaw Kutylowski, and Moti Yung. Hard Invalidation of Electronic Signatures. In Javier Lopez and Yongdong Wu, editors, *Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings*, volume 9065 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2015. (cited on page 168).

[HLP15] Carmit Hazay, Yehuda Lindell, and Arpita Patra. Adaptively Secure Computation with Partial Erasures. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 291–300. ACM, 2015. (cited on page 136).

[HM04] Dennis Hofheinz and Jörn Müller-Quade. Universally Composable Commitments Using Random Oracles. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 58–76. Springer, 2004. (cited on page 28).

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers'*

Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings, volume 7178 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2012. (cited on page 104).

[Hof11] Dennis Hofheinz. Possibility and Impossibility Results for Selective Decommitments. *J. Cryptology*, 24(3):470–516, 2011. (cited on page 134).

[HOR15] Brett Hemenway, Rafail Ostrovsky, and Alon Rosen. Non-committing Encryption from Φ-hiding. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2015. (cited on page 136).

[HORR16] Brett Hemenway, Rafail Ostrovsky, Silas Richelson, and Alon Rosen. Adaptive Security with Quasi-Optimal Rate. In Kushilevitz and Malkin [KM16], pages 525–541. (cited on page 136).

[HP16] Felix Heuer and Bertram Poettering. Selective Opening Security from Simulatable Data Encapsulation. In Cheon and Takagi [CT16], pages 248–277. (cited on page 136).

[HP17] Carmit Hazay and Arpita Patra. Efficient One-Sided Adaptively Secure Computation. *J. Cryptology*, 30(1):321–371, 2017. (cited on page 136).

[HPS12] Focke Höhne, Henrich C. Pöhls, and Kai Samelin. Rechtsfolgen editierbarer Signaturen. *Datenschutz und Datensicherheit*, 36(7):485–491, 2012. (cited on pages 200 and 206).

[HPW15] Carmit Hazay, Arpita Patra, and Bogdan Warinschi. Selective Opening Security for Receivers. In Iwata and Cheon [IC15], pages 443–469. (cited on pages 96, 97, 135, 136, 137, 139, 145 and 146).

[HR06] Shai Halevi and Tal Rabin, editors. *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*. Springer, 2006. (cited on pages 247 and 266).

[HR14] Dennis Hofheinz and Andy Rupp. Standard versus Selective Opening Security: Separation and Equivalence Results. In Lindell [Lin14], pages 591–615. (cited on pages 134 and 136).

[HRW16] Dennis Hofheinz, Vanishree Rao, and Daniel Wichs. Standard Security Does Not Imply Indistinguishability Under Selective Opening. In Hirt and Smith [HS16], pages 121–145. (cited on pages 134 and 136).

[HS15] Dennis Hofheinz and Victor Shoup. GNUC: A New Universal Composability Framework. *J. Cryptology*, 28(3):423–508, 2015. (cited on page 24).

[HS16] Martin Hirt and Adam D. Smith, editors. *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, 2016. (cited on pages 258 and 259).

[HSBB16] Moritz Horsch, Mario Schlipf, Johannes Braun, and Johannes A. Buchmann. Password Requirements Markup Language. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I*, volume 9722 of *Lecture Notes in Computer Science*, pages 426–439. Springer, 2016. (cited on page 6).

[HSMZ05] Xinyi Huang, Willy Susilo, Yi Mu, and Futai Zhang. On the Security of Certificateless Signature Schemes from Asiacrypt 2003. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *Cryptology and Network Security, 4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005, Proceedings*, volume 3810 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2005. (cited on page 220).

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a Random Oracle: Full Domain Hash from Indistinguishability Obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 201–220. Springer, 2014. (cited on page 28).

[HUM13] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Polynomial Runtime and Composability. *J. Cryptology*, 26(3):375–441, 2013. (cited on page 24).

[IC15] Tetsu Iwata and Jung Hee Cheon, editors. *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*. Springer, 2015. (cited on pages 257 and 259).

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In Preneel [Pre00], pages 221–242. (cited on page 136).

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures (Extended Abstract). In Kaliski Jr. [Kal97], pages 150–164. (cited on pages 220 and 221).

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic Signature Schemes. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose,*

CA, USA, February 18-22, 2002, Proceedings, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2002. (cited on page 7).

[Joy03] Marc Joye, editor. *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*. Springer, 2003. (cited on pages 253 and 272).

[JRP04] Anil K. Jain, Arun Ross, and Salil Prabhakar. An introduction to biometric recognition. *IEEE Trans. Circuits Syst. Video Techn.*, 14(1):4–20, 2004. (cited on page 6).

[JSS04] William K. Josephson, Emin Gün Sirer, and Fred B. Schneider. Peer-to-Peer Authentication with a Distributed Single Sign-On Service. In Geoffrey M. Voelker and Scott Shenker, editors, *Peer-to-Peer Systems III, Third International Workshop, IPTPS 2004, La Jolla, CA, USA, February 26-27, 2004, Revised Selected Papers*, volume 3279 of *Lecture Notes in Computer Science*, pages 250–258. Springer, 2004. (cited on pages 34 and 35).

[JT09] Stanislaw Jarecki and Gene Tsudik, editors. *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, volume 5443 of *Lecture Notes in Computer Science*. Springer, 2009. (cited on pages 239, 248, 252 and 255).

[JWdV06] Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors. *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. ACM, 2006. (cited on pages 241 and 242).

[KAKP06] Anastasis Kounoudes, Anixi Antonakoudi, Vassilis Kekatos, and Philippos Peleties. Combined Speech Recognition and Speaker Verification over the Fixed and Mobile Telephone Networks. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition, and Applications, SPPRA 2006, February 15-17, 2006, Innsbruck, Austria*, pages 228–233. IASTED/ACTA Press, 2006. (cited on page 6).

[Kal97] Burton S. Kaliski Jr., editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997. (cited on pages 238, 251, 260, 266 and 273).

[Kat15] Jonathan Katz, editor. *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, volume 9020 of *Lecture Notes in Computer Science*. Springer, 2015. (cited on page 246).

[KB13] Ashish Kundu and Elisa Bertino. Privacy-preserving authentication of trees and graphs. *Int. J. Inf. Sec.*, 12(6):467–494, 2013. (cited on page 7).

[KC17] Boris Köpf and Steve Chong, editors. *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017. (cited on pages 247 and 250).

[Kia11] Aggelos Kiayias, editor. *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*. Springer, 2011. (cited on pages 263 and 265).

[Kie16] Franziskus Kiefer. *Advancements in Password-based Cryptography*. PhD thesis, University of Surrey, January 2016. (cited on page 6).

[Kil01] Joe Kilian, editor. *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*. Springer, 2001. (cited on pages 239, 240, 246 and 270).

[KK99] Oliver Kömmerling and Markus G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In Scott B. Guthery and Peter Honeyman, editors, *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. USENIX Association, 1999. (cited on pages 6 and 92).

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal Security Proofs for Full Domain Hash, Revisited. In Pointcheval and Johansson [PJ12], pages 537–553. (cited on page 20).

[KL06] Marek Klonowski and Anna Lauks. Extended Sanitizable Signatures. In Min Surp Rhee and Byoungcheon Lee, editors, *Information Security and Cryptology - ICISC 2006, 9th International Conference, Busan, Korea, November 30 - December 1, 2006, Proceedings*, volume 4296 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2006. (cited on page 200).

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. (cited on pages 12, 13, 15, 16 and 17).

[KM13] Neal Koblitz and Alfred Menezes. Another look at non-uniformity. *Groups Complexity Cryptology*, 5(2):117–139, 2013. (cited on page 25).

[KM14] Franziskus Kiefer and Mark Manulis. Distributed Smooth Projective Hashing and Its Application to Two-Server Password Authenticated Key Exchange. In Boureanu et al. [BOV14], pages 199–216. (cited on page 35).

[KM15] Neal Koblitz and Alfred J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptography*, 77(2-3):587–610, 2015. (cited on page 28).

[KM16] Eyal Kushilevitz and Tal Malkin, editors. *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings,*

Part I, volume 9562 of *Lecture Notes in Computer Science*. Springer, 2016. (cited on pages 254, 259 and 272).

[KMPR05] Eike Kiltz, Anton Mityagin, Saurabh Panjwani, and Barath Raghavan. Append-Only Signatures. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 434–445. Springer, 2005. (cited on page 7).

[KMTG12] Jonathan Katz, Philip D. MacKenzie, Gelareh Taban, and Virgil D. Gligor. Two-server password-only authenticated key exchange. *J. Comput. Syst. Sci.*, 78(2):651–669, 2012. (cited on page 35).

[KR00a] David P. Kormann and Aviel D. Rubin. Risks of the Passport single signon protocol. *Computer Networks*, 33(1-6):51–58, 2000. (cited on page 6).

[KR00b] Hugo Krawczyk and Tal Rabin. Chameleon Signatures. In Stephen Welke, editor, *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*. The Internet Society, 2000. (cited on pages 168 and 197).

[KSS15] Stephan Krenn, Kai Samelin, and Dieter Sommer. Stronger Security for Sanitizable Signatures. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers*, volume 9481 of *Lecture Notes in Computer Science*, pages 100–117. Springer, 2015. (cited on pages 200, 202, 207 and 220).

[KT08] Ralf Küsters and Max Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008*, pages 270–284. IEEE Computer Society, 2008. (cited on page 149).

[KT09] Ralf Küsters and Max Tuengerthal. Universally Composable Symmetric Encryption. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 293–307. IEEE Computer Society, 2009. (cited on page 149).

[KT11] Ralf Küsters and Max Tuengerthal. Ideal Key Derivation and Encryption in Simulation-Based Security. In Kiayias [Kia11], pages 161–179. (cited on page 149).

[Küs06] Ralf Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, pages 309–320. IEEE Computer Society, 2006. (cited on page 24).

[KZ08] Aggelos Kiayias and Hong-Sheng Zhou. Equivocal Blind Signatures and Adaptive UC-Security. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, volume 4948 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008. (cited on pages 94, 103 and 106).

[LCC06] Feiyu Lei, Wen Chen, and Kefei Chen. A Non-committing Encryption Scheme Based on Quadratic Residue. In Albert Levi, Erkay Savas, Hüsnü Yenigün, Selim Balcisoy, and Yücel Saygin, editors, *Computer and Information Sciences - ISCIS 2006, 21th International Symposium, Istanbul, Turkey, November 1-3, 2006, Proceedings*, volume 4263 of *Lecture Notes in Computer Science*, pages 972–980. Springer, 2006. (cited on page 136).

[Lin09] Yehuda Lindell. General Composition and Universal Composability in Secure Multiparty Computation. *J. Cryptology*, 22(3):395–428, 2009. (cited on page 58).

[Lin14] Yehuda Lindell, editor. *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*. Springer, 2014. (cited on pages 235 and 259).

[Lin17] Yehuda Lindell. How to Simulate It - A Tutorial on the Simulation Proof Technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography.*, pages 277–346. Springer International Publishing, 2017. (cited on page 10).

[LMPY16] Benoît Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. Practical "Signatures with Efficient Protocols" from Simple Assumptions. In Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang, editors, *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, pages 511–522. ACM, 2016. (cited on page 7).

[LPJY15] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. *Des. Codes Cryptography*, 77(2-3):441–477, 2015. (cited on page 7).

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym Systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999. (cited on page C-1).

[LT11] Javier Lopez and Gene Tsudik, editors. *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, 2011. (cited on pages 238, 254 and 268).

[LYL11] Xuejia Lai, Moti Yung, and Dongdai Lin, editors. *Information Security and Cryptology - 6th International Conference, Inscrypt 2010, Shanghai, China, October*

20-24, 2010, Revised Selected Papers, volume 6584 of *Lecture Notes in Computer Science*. Springer, 2011. (cited on pages 251, 257 and 272).

[LZCS16] Russell W. F. Lai, Tao Zhang, Sherman S. M. Chow, and Dominique Schröder. Efficient Sanitizable Signatures Without Random Oracles. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*, volume 9878 of *Lecture Notes in Computer Science*, pages 363–380. Springer, 2016. (cited on pages 168 and 206).

[MCC+04] Robert L. Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated Security: The Shibboleth Approach. *EDUCAUSE Quarterly*, 27(4):12–17, 1 2004. (cited on page 35).

[Mer87] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987. (cited on page 7).

[Mit16] Arno Mittelbach. *Random Oracles in the Standard Model*. PhD thesis, TU Darmstadt, October 2016. (cited on pages 27 and 28).

[MKV+13] Michelle L. Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 173–186. ACM, 2013. (cited on page 6).

[MMBK10] David L. Mills, Jim Martin, Jack Burbank, and William Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010. Updated by RFC 7822. (cited on page 223).

[MPP14] Nikos Mavrogiannopoulos, Andreas Pashalidis, and Bart Preneel. Toward a secure Kerberos key exchange with smart cards. *Int. J. Inf. Sec.*, 13(3):217–228, 2014. (cited on page 35).

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-Based Signatures. In Kiayias [Kia11], pages 376–392. (cited on page 7).

[MR03] Philip D. MacKenzie and Michael K. Reiter. Networked cryptographic devices resilient to capture. *Int. J. Inf. Sec.*, 2(1):1–20, 2003. (cited on page 95).

[MUO96] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy Signatures for Delegating Signing Operation. In Li Gong and Jacques Stearn, editors, *CCS '96,*

Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, March 14-16, 1996., pages 48–57. ACM, 1996. (cited on pages 7 and 220).

[MUS+16] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 175–191. USENIX Association, 2016. (cited on page 6).

[MvO07] Mohammad Mannan and Paul C. van Oorschot. Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security, 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago, February 12-16, 2007. Revised Selected Papers*, volume 4886 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2007. (cited on page 95).

[Nie02] Jesper Buus Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In Yung [Yun02], pages 111–126. (cited on pages 28, 96, 97, 133, 134, 135, 136, 138, 140, 145, 147, 149, 155, F-1 and H-1).

[NP97] Moni Naor and Benny Pinkas. Visual Authentication and Identification. In Kaliski Jr. [Kal97], pages 322–336. (cited on page 6).

[NSJ08] Peng Ning, Paul F. Syverson, and Somesh Jha, editors. *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*. ACM, 2008. (cited on pages 240 and 245).

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, 1994. (cited on pages 34 and 35).

[NY90] Moni Naor and Moti Yung. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 427–437. ACM, 1990. (cited on page 22).

[Oka06] Tatsuaki Okamoto. Efficient Blind and Partially Blind Signatures Without Random Oracles. In Halevi and Rabin [HR06], pages 80–99. (cited on page 222).

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes*

in Computer Science, pages 223–238. Springer, 1999. (cited on pages 85, 147 and D-1).

[Ped91] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991. (cited on pages 24 and 183).

[Pie12] Krzysztof Pietrzak. Cryptography from Learning Parity with Noise. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2012. (cited on page 8).

[PJ12] David Pointcheval and Thomas Johansson, editors. *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012. (cited on pages 238 and 262).

[PM03] Andreas Pashalidis and Chris J. Mitchell. A Taxonomy of Single Sign-On Systems. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, volume 2727 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2003. (cited on page 35).

[PPS+13] Henrich C. Pöhls, Stefan Peters, Kai Samelin, Joachim Posegga, and Hermann de Meer. Malleable Signatures for Resource Constrained Platforms. In Lorenzo Cavallaro and Dieter Gollmann, editors, *Information Security Theory and Practice. Security of Mobile and Cyber-Physical Systems, 7th IFIP WG 11.2 International Workshop, WISTP 2013, Heraklion, Greece, May 28-30, 2013. Proceedings*, volume 7886 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2013. (cited on pages 188 and 200).

[Pre00] Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000. (cited on pages 243 and 260).

[PS14] Henrich C. Pöhls and Kai Samelin. On Updatable Redactable Signatures. In Boureanu et al. [BOV14], pages 457–475. (cited on pages 7 and 220).

[PS15] Henrich C. Pöhls and Kai Samelin. Accountable Redactable Signatures. In *10th International Conference on Availability, Reliability and Security, ARES 2015,*

Toulouse, France, August 24-27, 2015, pages 60–69. IEEE Computer Society, 2015. (cited on page 200).

[PSP11] Henrich C. Pöhls, Kai Samelin, and Joachim Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In Lopez and Tsudik [LT11], pages 166–182. (cited on page 200).

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2005. (cited on page 135).

[PW01] Birgit Pfitzmann and Michael Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pages 184–200. IEEE Computer Society, 2001. (cited on page 24).

[PW03] Birgit Pfitzmann and Michael Waidner. Analysis of Liberty Single-Sign-on with Enabled Clients. *IEEE Internet Computing*, 7(6):38–44, 2003. (cited on page 35).

[QQQ+89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to Explain Zero-Knowledge Protocols to Your Children. In Brassard [Bra90], pages 628–631. (cited on page 8).

[Rab98] Tal Rabin. A Simplified Approach to Threshold and Proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 1998. (cited on page 94).

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010. (cited on pages 236 and 245).

[RAB14] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. (cited on page 6).

[RCB01] Nalini K. Ratha, Jonathan H. Connell, and Ruud M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001. (cited on page 6).

[RCV14] Cristina Romero-Tris, Jordi Castellà-Roca, and Alexandre Viejo. Distributed system for private web search with untrusted partners. *Computer Networks*, 67:26–42, 2014. (cited on page A-2).

[Reg04] Oded Regev. Quantum Computation and Lattice Problems. *SIAM J. Comput.*, 33(3):738–760, 2004. (cited on page 8).

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009. (cited on page 8).

[Riv98] Ronald L. Rivest. Can We Eliminate Certificate Revocations Lists? In Rafael Hirschfeld, editor, *Financial Cryptography, Second International Conference, FC'98, Anguilla, British West Indies, February 23-25, 1998, Proceedings*, volume 1465 of *Lecture Notes in Computer Science*, pages 178–183. Springer, 1998. (cited on page 219).

[RMS08] Qiong Ren, Yi Mu, and Willy Susilo. Mitigating Phishing with ID-based Online/Offline Authentication. In Ljiljana Brankovic and Mirka Miller, editors, *Sixth Australasian Information Security Conference, AISC 2008, Wollongong, NSW, Australia, January 2008*, volume 81 of *CRPIT*, pages 59–64. Australian Computer Society, 2008. (cited on pages 168 and 169).

[Rog06] Phillip Rogaway. Formalizing Human Ignorance. In Phong Q. Nguyen, editor, *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006. (cited on page 15).

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011. (cited on pages 236, 239 and 240).

[RR06] David Recordon and Drummond Reed. OpenID 2.0: a platform for user-centric identity management. In Ari Juels, Marianne Winslett, and Atsuhiro Goto, editors, *Proceedings of the 2006 Workshop on Digital Identity Management, Alexandria, VA, USA, November 3, 2006*, pages 11–16. ACM, 2006. (cited on page 35).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. (cited on pages 7, 8, 14 and 18).

[RY07] Thomas Ristenpart and Scott Yilek. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2007. (cited on page 197).

[SBP16] Alexander Spenke, Ralph Breithaupt, and Rainer Plaga. An Arbiter PUF Secured by Remote Random Reconfigurations of an FPGA. In Franz and Papadimitratos [FP16], pages 140–158. (cited on page 6).

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content Extraction Signatures. In Kwangjo Kim, editor, *Information Security and Cryptology - ICISC 2001, 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2001. (cited on page 7).

[Sch91] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptology*, 4(3):161–174, 1991. (cited on pages 82, 83 and 135).

[Sch05] Bruce Schneier. Two-factor authentication: too little, too late. *Commun. ACM*, 48(4):136, 2005. (cited on page 6).

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust Non-interactive Zero Knowledge. In Kilian [Kil01], pages 566–598. (cited on page 40).

[SG02] Victor Shoup and Rosario Gennaro. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. *J. Cryptology*, 15(2):75–96, 2002. (cited on page 146).

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979. (cited on pages 41 and 85).

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004. (cited on page 9).

[Sho05] Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005. (cited on pages 239 and 249).

[SNS88] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the USENIX Winter Conference. Dallas, Texas, USA, January 1988*, pages 191–202. USENIX Association, 1988. (cited on page 35).

[SR10] Daniel Slamanig and Stefan Rass. Generalizations and Extensions of Redactable Signatures with Applications to Electronic Healthcare. In Bart De Decker and Ingrid Schaumüller-Bichl, editors, *Communications and Multimedia Security, 11th IFIP TC 6/TC 11 International Conference, CMS 2010, Linz, Austria, May 31 - June 2, 2010. Proceedings*, volume 6109 of *Lecture Notes in Computer Science*, pages 201–213. Springer, 2010. (cited on page 7).

[SS96] Ravi S. Sandhu and Pierangela Samarati. Authentication, Access Control, and Audit. *ACM Comput. Surv.*, 28(1):241–243, 1996. (cited on page 6).

[SS13] Kazue Sako and Palash Sarkar, editors. *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*. Springer, 2013. (cited on page 237).

[SS17] Andrei Sabelfeld and Matthew Smith, editors. *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017. (cited on pages 236 and 237).

[TDB16] Giulia Traverso, Denise Demirel, and Johannes A. Buchmann. *Homomorphic Signature Schemes - A Survey*. Springer Briefs in Computer Science. Springer, 2016. (cited on page 200).

[USB+15] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 463–481. USENIX Association, 2015. (cited on page 6).

[Vau06] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006. (cited on pages 235 and 244).

[Vel12] Milen V. Velev. Relativistic mechanics in multiple time dimensions. *Physics Essays*, 25(3):403–438, 2012. (cited on page 44).

[Ven14] Muthuramakrishnan Venkitasubramaniam. On Adaptively Secure Protocols. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 455–475. Springer, 2014. (cited on page 94).

[VWG07] Dennis Vermoen, Marc F. Witteman, and Georgi Gaydadjiev. Reverse Engineering Java Card Applets Using Power Analysis. In Damien Sauveron, Constantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, First IFIP TC6 / WG 8.8 / WG 11.2 International Workshop, WISTP 2007, Heraklion, Crete, Greece, May 9-11, 2007, Proceedings*, volume 4462 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2007. (cited on page 6).

[WD95] Ira S. Winkler and Brian Dealy. Information Security Technology? Don't Rely on It. A Case Study in Social Engineering. In Frederick M. Avolio and Steven M.

Bellovin, editors, *Proceedings of the 5th USENIX Security Symposium, Salt Lake City, Utah, USA, June 5-7, 1995*. USENIX Association, 1995. (cited on page 6).

[Wik16] Douglas Wikström. Simplified Universal Composability Framework. In Kushilevitz and Malkin [KM16], pages 566–595. (cited on page 24).

[Wor08] Michael Workman. A test of interventions for security threats from social engineering. *Inf. Manag. Comput. Security*, 16(5):463–483, 2008. (cited on page 6).

[WS12] Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012. (cited on pages 236 and 243).

[Wu99] Thomas D. Wu. A Real-World Analysis of Kerberos Password Security. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*. The Internet Society, 1999. (cited on page 35).

[WYX13] Guilin Wang, Jiangshan Yu, and Qi Xie. Security Analysis of a Single Sign-On Mechanism for Distributed Computer Networks. *IEEE Trans. Industrial Informatics*, 9(1):294–302, 2013. (cited on pages 34 and 35).

[XS03] Shouhuai Xu and Ravi S. Sandhu. Two Efficient and Provably Secure Schemes for Server-Assisted Threshold Signatures. In Joye [Joy03], pages 355–372. (cited on page 95).

[XSA+16] Wenjie Xiong, André Schaller, Nikolaos Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. Run-Time Accessible DRAM PUFs in Commodity Devices. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 432–453. Springer, 2016. (cited on page 6).

[YSL10] Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. Trapdoor Sanitizable Signatures Made Easy. In Zhou and Yung [ZY10], pages 53–68. (cited on page 200).

[Yun02] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002. (cited on pages 235, 249 and 266).

[ZB10] Huafei Zhu and Feng Bao. Error-free, Multi-bit Non-committing Encryption with Constant Round Complexity. In Lai et al. [LYL11], pages 52–61. (cited on page 136).

[Zhe97] Yuliang Zheng. Digital Signcryption or How to Achieve Cost(Signature & Encryption) << Cost(Signature) + Cost(Encryption). In Kaliski Jr. [Kal97], pages 165–179. (cited on page 161).

[ZLZL09] Shangping Zhong, Xiangwen Liao, Xue Zhang, and Jingqu Lin. A Novel Distributed Single Sign-On Scheme with Dynamically Changed Threshold Value. In *Proceedings of the Fifth International Conference on Information Assurance and Security, IAS 2009, Xi'An, China, 18-20 August 2009*, pages 563–566. IEEE Computer Society, 2009. (cited on pages 34 and 35).

[ZP16] Vassilis Zikas and Roberto De Prisco, editors. *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, volume 9841 of *Lecture Notes in Computer Science*. Springer, 2016. (cited on pages 246, 250, 254 and 255).

[ZSS03] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. *IACR Cryptology ePrint Archive*, 2003:208, 2003. (cited on page 168).

[ZY10] Jianying Zhou and Moti Yung, editors. *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *Lecture Notes in Computer Science*, 2010. (cited on pages 237 and 272).

[ZZ04] Kan Zhang and Yuliang Zheng, editors. *Information Security, 7th International Conference, ISC 2004, Palo Alto, CA, USA, September 27-29, 2004, Proceedings*, volume 3225 of *Lecture Notes in Computer Science*. Springer, 2004. (cited on pages 240 and 251).

Appendix A

Instantiation of TEnc

Now is presented how suitable TEncs can be constructed. First is shown how a general t-out-of-n version can be realized, while a more efficient n-out-of-n is given afterwards.

A.1 A Generalized t-out-of-n TEnc

This is essentially the construction given by Camenisch et al. [CLLN14], based on existing work [DF89], but altered for the used notation, while the decryption also takes a label used for the underlying proof system.

Let $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$ be a group generator for a prime-order, and multiplicatively written, group $\mathbb{G}$[1], along with a generator g of order q of that group, such that $\langle g \rangle = \mathbb{G}$, and that the decisional Diffie-Hellman assumption holds.

Construction A.1 (TEnc). *A suitable construction is given as follows, where the message space $\mathcal{MS}$ is thus a group $\mathbb{G}$.*

$\mathsf{PPGen}_{\mathsf{TEnc}}$. *To generate the public parameters, do:*

1. *Generate* $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$, *where DDH holds in* $\mathbb{G}$.
2. *Generate* $\mathsf{crs}_{\mathsf{NIZKPoK}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NIZKPoK}}(1^\lambda, L)$.
3. *Return* $\mathsf{pp}_{\mathsf{TEnc}} = (\mathsf{crs}_{\mathsf{NIZKPoK}}, \mathbb{G}, g, q)$.

$\mathsf{KeyGen}_{\mathsf{TEnc}}$. *To generate the public key, and the corresponding partial key pairs, do:*

1. *Pick* $t+1$ *coefficients, i.e.,* $(a_0, a_1, \ldots, a_t) \xleftarrow{\$} (\mathbb{Z}_q^*)^t$. *Let* $p(x) := \sum_{k=0}^{t} a_k x^k \pmod q$ *be the polynomial defined by these coefficients.*
2. *For all* $i \in \{1, 2, \ldots, n\}$, *let* $\mathsf{sk}^i_{\mathsf{PTEnc}} \leftarrow p(i)$, $\mathsf{pk}^i_{\mathsf{PTEnc}} \leftarrow g^{\mathsf{sk}^i_{\mathsf{PTEnc}}}$, *and* $\mathsf{pk}_{\mathsf{TEnc}} \leftarrow g^{a_0} = g^{p(0)}$.
3. *Return* $(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}^i_{\mathsf{PTEnc}}, \mathsf{pk}^i_{\mathsf{PTEnc}})_{1 \le i \le n})$.

$\mathsf{Enc}_{\mathsf{TEnc}}$. *To encrypt a message* $m \in \mathbb{G}$, *do:*

1. *Pick a random* $r \xleftarrow{\$} \mathbb{Z}_q^*$.

[1] Clearly, DLGen only returns a description of $\mathbb{G}$.

2. *Return* $(g^r, \mathsf{pk}_{\mathsf{TEnc}}^r m)$.

$\mathsf{PDec}_{\mathsf{TEnc}}$. *To partially decrypt a ciphertext* $c = (u, v)$ *w.r.t. to label* ℓ*, do:*

1. *Let* $d_i \leftarrow u^{\mathsf{sk}_{\mathsf{PTEnc}}^i}$
2. *Let* $\pi_i \xleftarrow{\$} \mathsf{Prove}_{\mathsf{NIZKPoK}}\{(\mathsf{sk}_{\mathsf{PTEnc}}^i) : d_i = u^{\mathsf{sk}_{\mathsf{PTEnc}}^i} \wedge \mathsf{pk}_{\mathsf{PTEnc}}^i = g^{\mathsf{sk}_{\mathsf{PTEnc}}^i}\}(\mathsf{pk}_{\mathsf{TEnc}}, c, \mathsf{pk}_{\mathsf{PTEnc}}^i, \ell)$
3. *Return* (d_i, π_i).

$\mathsf{VfDec}_{\mathsf{TEnc}}$. *To verify a decryption share* (d_i, π_i) *w.r.t.* ℓ *and* (u, v)*, do:*

1. *Return* true*, if* π_i *is valid, and* false *otherwise.*

$\mathsf{Dec}_{\mathsf{TEnc}}$. *To complete the decryption of* $c = (u, v)$*, on input of* $(d_i)_{0<i\leq t+1}$ *do:*

1. *Let* $I = \{i_1, i_2, \ldots, i_{t+1}\}$ *be the index set corresponding to where the polynomial was evaluated at key generation w.r.t.* $(d_i, \pi_i)_{0<i\leq t+1}$.
2. *Express* $p(0)$ *as a linear combination of* $p(i_1), p(i_2), \ldots, p(i_{t+1})$ *using standard polynomial interpolation techniques using Lagrange, i.e.,* $p(0) = \sum_{k=1}^{t+1} u_k p(i_k)$.
3. *Let* $w \leftarrow \prod_{k=1}^{t+1} d_{i_k}^{u_k}$.
4. *Return* vw^{-1}.

That the above construction is secure was already proven by Camenisch et al. [CLLN14], with the exception of the additional label, which can be ignored in the proof of security.

Note, checking whether a $\mathsf{pk}_{\mathsf{PTEnc}}^i$ belongs to a $\mathsf{sk}_{\mathsf{PTEnc}}^i$ is trivial, i.e., by performing a simple exponentiation, while π_i only proves the equivalence of discrete logarithms.

A.2 A Simplified n-out-of-n TEnc

The above construction is for the general t-out-of-n case. If, as required for the single sign-on protocols, resorts to the special n-out-of-n case, a conceptually simpler construction exists, which shares the exponents in an additive manner. This saves reconstructing the polynomial and is thus faster.

The given construction is derived from Romero-Tris et al. [RCV14], but adjusted for the used notation.

Construction A.2 (TEnc)**.** *Consider the following construction, where* $\mathcal{MS}$ *is as before.*

$\mathsf{PPGen}_{\mathsf{TEnc}}$. *To generate the public parameters, do:*

1. *As before.*

$\mathsf{KeyGen}_{\mathsf{TEnc}}$. *To generate the public key, and the corresponding partial key pairs, do:*

1. *Choose* n *random exponents* $\mathsf{sk}_{\mathsf{PTEnc}}^i \xleftarrow{\$} \mathbb{Z}_q^*$.
2. *For all* $i \in \{1, 2, \ldots, n\}$*, let* $\mathsf{pk}_{\mathsf{PTEnc}}^i \leftarrow g^{\mathsf{sk}_{\mathsf{PTEnc}}^i}$*, and* $\mathsf{pk}_{\mathsf{TEnc}} \leftarrow g^{\sum \mathsf{sk}_{\mathsf{PTEnc}}^i}$.
3. *Return* $(\mathsf{pk}_{\mathsf{TEnc}}, (\mathsf{sk}_{\mathsf{PTEnc}}^i, \mathsf{pk}_{\mathsf{PTEnc}}^i)_{1\leq i\leq n})$.

$\mathsf{Enc}_{\mathsf{TEnc}}$. *To encrypt a message $m \in \mathbb{G}$, do:*

1. *As before.*

$\mathsf{PDec}_{\mathsf{TEnc}}$. *To partially decrypt a ciphertext $c = (u, v)$ w.r.t. to label ℓ, do:*

1. *As before.*

$\mathsf{VfDec}_{\mathsf{TEnc}}$. *To verify a decryption share (d_i, π_i) w.r.t. ℓ and (u, v), do:*

1. *As before.*

$\mathsf{Dec}_{\mathsf{TEnc}}$. *To complete the decryption of $c = (u, v)$, on input of $(d_i)_{0<i\leq n}$ do:*

1. *Let $d \leftarrow \prod d_i$.*
2. *Return vd^{-1}.*

Appendix B

Groth's Structure Preserving Signature Scheme

In the second construction of SSO, a signature to issue tokens to user that allows for efficient proofs of knowledge of a signature as well as efficient verifiable encryption of the signature and the messages signed is required. To be able to use ElGamal encryption for the latter, i.e., to avoid the more expensive encryption of exponents, one can use a recent signature scheme by Groth [Gro15], where signatures and messages are all group elements. This scheme is recalled in the following, which is secure in the generic-group model.

B.1 Construction

Groth defines the scheme to sign a matrix of group elements. In this case, one can consider the special case where only a vector of n group elements is signed. The signature scheme assumes the availability of system parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, x)$, where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q)$ define a bilinear maps setting and $x \xleftarrow{\$} \mathbb{G}_1$ is an additional random group element.

Construction B.1 (DSIG)**.** *The signature scheme can now be constructed as follows.*

$\mathsf{PPGen}_{\mathsf{DSIG}}$**.** *To generate the public parameters with input n, do:*

1. *Generate* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda)$
2. *Choose* $x \xleftarrow{\$} \mathbb{G}_1$.
3. *Return* $\mathsf{pp}_{\mathsf{DSIG}} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, x)$.

$\mathsf{KeyGen}_{\mathsf{Sig}}$**.** *To generate a key pair, do:*

1. *Choose* $v \xleftarrow{\$} \mathbb{Z}_q^*$.
2. *For each* $i \in [1, n-1]$*, draw* $w_i \xleftarrow{\$} \mathbb{Z}_q^*$.
3. *Let* $z_i \leftarrow g_2^{w_i}$.
4. *Compute* $y \leftarrow g_2^v$.
5. *Return* $(((w_i)_{i\in[1,n-1]}, v), ((z_i)_{i\in[1,n-1]}, y))$.

$\mathsf{Sign}_{\mathsf{Sig}}$. *On input a message* $(m_1, \ldots, m_n) \in \mathbb{G}_1^n$ *and secret key* $\mathsf{sk}_{\mathsf{Sig}} = ((w_i)_{i\in[1,n-1]}, v)$, *do:*

1. *Choose a random* $u \xleftarrow{\$} \mathbb{Z}_q^*$.
2. *Let* $r \leftarrow g_2^u$.
3. *Let* $s \leftarrow (xg_1^v)^{1/u}$.
4. *Let* $t \leftarrow (\prod_{i=1}^{n-1} m_i^{w_i}(m_n x^v))^{1/u}$.
5. *Return* $\sigma = (r, s, t)$.

$\mathsf{Verify}_{\mathsf{Sig}}$. *To verify a signature, do:*

1. *Parse input as* $\sigma = (r, s, t)$ *and* $\mathsf{pk}_{\mathsf{Sig}} = ((z_i)_{i\in[1,n-1]}, y)$.
2. *If* $m_i, s, t \notin \mathbb{G}_1$, *return* false.
3. *If* $r \notin \mathbb{G}_2$, *return* false.
4. *If* $e(s, r) = e(g_1, y) \cdot e(x, g_2) \wedge e(t, r) = e(m_n, g_2) \prod_{i=1}^{n-1} e(m_i, z_i) \cdot e(x, y)$, *return* true.
5. *Return* false.

B.2 Zero-Knowledge Proofs

As pointed out by Groth, a signature $\sigma = (r, s, t)$ can be randomized to obtain a signature $\sigma' = (r', s', t')$ by picking a random $u' \xleftarrow{\$} \mathbb{Z}_q^*$ and computing: $r' \leftarrow r^{u'}$, $s' \leftarrow s^{1/u'}$, and $t' \leftarrow t^{1/u'}$.

Then, a proof of knowledge of a signature on messages m_i that is online extractable w.r.t. a $n + 2$ ElGamal public keys $(\tilde{y}_s, \tilde{y}_1, \ldots, \tilde{y}_n, \tilde{y}_t)$, where the base for these keys is g_1, contained in the CRS is done as follows.

- Encrypt the witnesses by selecting $\bar{r} \xleftarrow{\$} \mathbb{Z}_q^*$ and computing $\bar{g} = g^{\bar{r}}$ and:

$$\bar{s} = s'\tilde{y}_s^{\bar{r}}, \quad \bar{m}_i = m_i\tilde{y}_i^{\bar{r}}, \quad \bar{t} = t'\tilde{y}_t{}^{\bar{r}}$$

- Compute the following proof:

$$\pi_\sigma \xleftarrow{\$} \mathsf{SPK}\Big[(\bar{r}) : e(g_1, y) \cdot e(x, g_2)/e(\bar{s}, r') = e(\tilde{y}_s, r')^{-\bar{r}} \ \wedge \ \bar{g} = g^{\bar{r}} \ \wedge \\ \frac{e(\bar{t}, r')}{e(x, y) \cdot e(\bar{m}_n, g_2) \cdot \prod_{i=1}^{n-1} e(\bar{m}_i, z_i)} = \Big(\frac{e(\tilde{y}_t, r')}{e(\tilde{y}_n, g_2) \cdot \prod_{i=1}^{n-1} e(\tilde{y}_i, z_i)}\Big)^{\bar{r}}\Big](r', \bar{s}, \bar{t}, \bar{m}_1, \ldots, \bar{m}_n)$$

The overall on-line extractable proof of knowledge of a signature consists of $(r', \bar{s}, \bar{t}, \bar{m}_1, \bar{m}_2, \ldots, \bar{m}_n, \pi_\sigma)$.

From the first term one can derive $e(\bar{s}\tilde{y}_s^{-\bar{r}}, r') = e(g_1, y) \cdot e(x, g_2)$ and from the third term $e(\bar{t}\tilde{y}_t^{-\bar{r}}, r') = e(x, y) \cdot e(\bar{m}_n\tilde{y}_n^{-\bar{r}}, g_2) \cdot \prod_{i=1}^{n-1} e(\bar{m}_i\tilde{y}_i^{-\bar{r}}, z_i)$. In other words, this means that $(r', \bar{s}\tilde{y}_s^{-\bar{r}}, \bar{t}\tilde{y}_t^{-\bar{r}})$ is a signature on messages $\bar{m}_1\tilde{y}_1^{-\bar{r}}$, …, $\bar{m}_n\tilde{y}_n^{-\bar{r}}$. While $\bar{r}$ is only known to the prover, it is ensured by the second term $\bar{g} = g^{\bar{r}}$ in π_σ that the signature and messages can be computed if one is privy to the discrete logs of the ElGamal public keys $(\tilde{y}_s, \tilde{y}_1, \ldots, \tilde{y}_n, \tilde{y}_t)$ w.r.t. base g_1.

In the generalized case, i.e., where some elements are public knowledge and encryptions are used across multiple terms, the corresponding proofs can be implemented as follows.

A proof of knowledge of a signature on messages m_i that is online extractable w.r.t. a $n+2$ ElGamal public keys $(\tilde{y}_s, \tilde{y}_1, \ldots, \tilde{y}_n, \tilde{y}_t)$, where the base for these keys is g_1, contained in the CRS is done as follows.

- Encrypt the witnesses by selecting $\bar{r} \xleftarrow{\$} \mathbb{Z}_q^*$, and $\bar{r}_i \xleftarrow{\$} \mathbb{Z}_q^*$, where m_i is not to be made public.
- Compute $\bar{g} = g^{\bar{r}}$. For simplicity, it is assumed that m_n is always hidden, while $\mathcal{I}$ is the index set of the non-hidden elements. Finally compute:

$$\bar{s} = s'\tilde{y}_s^{\bar{r}}, \quad \bar{m}_i = m_i\tilde{y}_i^{\bar{r}} \text{ (for each hidden } m_i), \quad \bar{t} = t'\tilde{y}_t{}^{\bar{r}}$$

- Compute the proof

$$\pi_\sigma \xleftarrow{\$} \mathsf{SPK}\Big[(\bar{r}) : e(g_1, y) \cdot e(x, g_2)/e(\bar{s}, r') = e(\tilde{y}_s, r')^{-\bar{r}} \ \wedge \ \bar{g} = g^{\bar{r}} \ \wedge \\ \frac{e(\bar{t}, r')}{e(x, y) \cdot e(\bar{m}_n, g_2) \cdot \prod_{i \notin \mathcal{I}, i \neq n} e(\bar{m}_i, z_i) \cdot \prod_{i \in \mathcal{I}} e(m_i, z_i)} = \\ \Big(\frac{e(\tilde{y}_t, r')}{e(\tilde{y}_n, g_2) \cdot \prod_{i \notin \mathcal{I}, i \neq n} e(\tilde{y}_i, z_i)}\Big)^{\bar{r}}\Big](r', \bar{s}, \bar{t}, (\bar{m}_i)_{i \notin \mathcal{I}})$$

It is noted that when the computations of π_σ are implemented, there are a number of optimizations possible in terms of performing exponentiations in $\mathbb{G}_1$ and then compute pairing instead of doing the exponentiations in $\mathbb{G}_T$. Also, the number of pairing computations can be reduced by suitably combining pairings that have the same value as second argument.

Appendix C

Camenisch et al.'s Pseudonym System

This chapter shows how one can construct a scope-exclusive pseudonym-system.

C.1 Pseudonym Systems

Users can be known under different pseudonyms to service providers in the second protocol presented. This protocol uses a - less strict - version of the definitions given by Camenisch et al. [CKL+15], i.e., only collision-resistance and unlinkability [LRSW99] are defined, but not key-extractability [CKL+15].

Definition C.1 (Pseudonym Systems)**.** *A pseudonym system* NYM *consists of the algorithms* $\{\mathsf{PPGen}_{\mathsf{NYM}}, \mathsf{KeyGen}_{\mathsf{NYM}}, \mathsf{Gen}_{\mathsf{NYM}}\}$ *such that:*

$\mathsf{PPGen}_{\mathsf{NYM}}$**.** *This algorithm outputs parameters:*

$$\mathsf{pp}_{\mathsf{NYM}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NYM}}(1^{\lambda})$$

The public parameters are assumed to be input to all following algorithms.

$\mathsf{KeyGen}_{\mathsf{NYM}}$**.** *A user generates his secret key:*

$$(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{NYM}}(\mathsf{pp}_{\mathsf{NYM}})$$

$\mathsf{Present}_{\mathsf{NYM}}$**.** *A pseudonym* nym *is deterministically generated as follows:*

$$\mathsf{nym} \leftarrow \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}, \mathsf{scope})$$

Correctness. For each pseudonym system NYM, the usual correctness properties are required to hold. In particular, it is required that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{NYM}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NYM}}(1^{\lambda})$, for all $(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{NYM}}(\mathsf{pp}_{\mathsf{NYM}})$, for all $\mathsf{scope} \in \{0,1\}^*$, for all $\mathsf{nym} \leftarrow \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}, \mathsf{scope})$, $\mathsf{nym} \neq \perp$. This definition captures perfect correctness.

C.2 Security

Additionally, some security requirements are necessary, presented next.

Experiment $\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{NYM}}(\lambda)$:
$\mathsf{pp}_{\mathsf{NYM}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{NYM}}(1^\lambda)$
$(\mathsf{usk}_0, \mathsf{upk}_0) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{NYM}}(\mathsf{pp}_{\mathsf{NYM}})$
$(\mathsf{usk}_1, \mathsf{upk}_1) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{NYM}}(\mathsf{pp}_{\mathsf{NYM}})$
$b \xleftarrow{\$} \{0,1\}$
$L \leftarrow \emptyset$
$(\mathsf{scope}^*, state_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}_0(\mathsf{usk}_0,\cdot),\mathcal{O}_1(\mathsf{usk}_1,\cdot)}(\mathsf{pp}_{\mathsf{NYM}}, \mathsf{upk}_0, \mathsf{upk}_1)$
where oracle $\mathcal{O}_i$, for $i = 0, 1$, on input scope:
$L \leftarrow L \cup \{\mathsf{scope}\}$
return $\mathsf{nym} \xleftarrow{\$} \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}_i, \mathsf{scope})$
$\mathsf{nym}^* \xleftarrow{\$} \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}_b, \mathsf{scope}^*)$
$b' \xleftarrow{\$} \mathcal{A}_2^{\mathcal{O}_0(\mathsf{usk}_0,\cdot),\mathcal{O}_1(\mathsf{usk}_1,\cdot)}(state_{\mathcal{A}}, \mathsf{nym}^*)$
return $b' \xleftarrow{\$} \{0,1\}$, if $\mathsf{scope}^* \in L$
return 1, if $b = b'$
return 0

Figure C.1: NYM Unlinkability

Collision Resistance. Collision resistance guarantees that for each scope string, any two users will have different pseudonyms with overwhelming probability.

Definition C.2 (Collision Resistance)**.** *A pseudonym system is* collision resistant*, if for every PPT algorithm $\mathcal{A}$ there is a negligible function ν such that:*

$$\Pr[\mathsf{nym}_0 = \mathsf{nym}_1 \ \wedge \ \mathsf{nym}_0 \neq \perp \ \wedge \ \mathsf{usk}_0 \neq \mathsf{usk}_1 : \mathsf{nym}_0 \leftarrow \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}_0, \mathsf{scope}),$$
$$\mathsf{nym}_1 \leftarrow \mathsf{Gen}_{\mathsf{NYM}}(\mathsf{usk}_1, \mathsf{scope}), (\mathsf{usk}_0, \mathsf{usk}_1, \mathsf{scope}) \xleftarrow{\$} \mathcal{A}(\mathsf{PPGen}_{\mathsf{NYM}}(1^\lambda))] \leq \nu(\lambda)$$

Unlinkability. To be useable in a broader setting, a NYM also needs to be unlinkable. Unlinkability guarantees that users cannot be traced across different scopes [CKL+15].

Definition C.3 (Unlinkability)**.** *A pseudonym system* NYM *is unlinkable, if for every PPT adversary $\mathcal{A}$ there is a negligible function ν such that:*

$$\left|\Pr[\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{NYM}}(1^\lambda) = 1] - \tfrac{1}{2}\right| \leq \nu(\lambda)$$

The corresponding experiment is depicted in Figure C.1.

C.3 Construction

Construction C.4 (NYM)**.** *A suitable pseudonym-system can be constructed as follows, as presented by Camenisch et al. [CKL+15], which also provide a security proof.*

$\mathsf{PPGen}_{\mathsf{NYM}}$**.** *To generate the public parameters, do:*

1. *Generate* $(\mathbb{G}, g, q) \xleftarrow{\$} \mathsf{DLGen}(1^\lambda)$*, where DDH holds in* $\mathbb{G}$*.*
2. *Return* $\mathsf{pp}_{\mathsf{NYM}} = (\mathbb{G}, g, q)$*.*

$\mathsf{KeyGen}_{\mathsf{NYM}}$. *To generate key pair, do:*

1. *Choose* $\mathsf{usk} \xleftarrow{\$} \mathbb{Z}_q^*$*.*
2. *Let* $\mathsf{upk} \leftarrow g^{\mathsf{usk}}$*.*
3. *Return* $(\mathsf{usk}, \mathsf{upk})$*.*

$\mathsf{Gen}_{\mathsf{NYM}}$. *To generate a pseudonym, do:*

1. *Return* $\mathsf{nym} \leftarrow (\mathcal{H}(\mathsf{scope}))^{\mathsf{usk}}$*, where* $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$ *denotes a random oracle.*

Note, however, that in the original construction g^{usk} is not given away as upk. However, as g is a random generator and the probability that the adversary finds an input which maps an input to g, giving out upk as some valid pseudonym only negligibly changes the adversary's view.

Appendix D

A Pseudonym-System from Bilinear Maps

In the following, a pseudonym-system where the secret key is group-element and not a exponent is presented. It is based on the ideas given by Camenisch et al. [CKL+15]. See also Appendix C.

D.1 Decisional Co-Diffie-Hellman Assumption

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda)$ be defined as usual. The Decisional Co-Bilinear DH-Assumption now states that given g_1^x, g_2^y and g_2^z, no PPT adversary $\mathcal{A}$ can distinguish whether $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q$ [BLS01].

More formally, there is the following definition:

Definition D.1 (Decisional Co-Diffie-Hellman Assumption)**.** *The Decisional Co-Diffie-Hellman Assumption holds, if for every PPT adversary $\mathcal{A}$ there is a negligible function ν such that:*

$$\Big|\Pr[a = b : x \xleftarrow{\$} \mathbb{Z}_q, y \xleftarrow{\$} \mathbb{Z}_q, z_0 \leftarrow xy, z_1 \xleftarrow{\$} \mathbb{Z}_q, \\ b \xleftarrow{\$} \{0,1\}, (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda) \\ a \xleftarrow{\$} \mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, g_1^x, g_2^y, g_2^{z_b})] - \tfrac{1}{2}\Big| \leq \nu(\lambda)$$

D.2 Construction

Now is shown how to construct such a scheme in a setting of bilinear maps. The scheme is derived from the one presented by Camenisch et al. [CKL+15] (See also Appendix C). However, their secret key is some exponent, which requires an expensive encryption scheme to be online-extractable. In the bilinear setting, this can be avoided by using a group element as the secret key, and using standard ElGamal encryption [Gam84], which is by far less demanding than Paillier [Pai99] or Camenisch-Shoup [CS03]. The resulting scheme in then also trivially key-extractable [CKL+15].

One may ask why there is still a random-oracles and no (fully) structure-preserving primitive is presented. However, as shown by Abe et al. [ACDD14], this cannot be achieved, as - by their very definition - pseudonyms are deterministic.

Construction D.2 (NYM). *A suitable pseudonym-system can be constructed as follows:*

$\mathsf{PPGen}_{\mathsf{NYM}}$. *To generate the public parameters, do:*

1. *Return* $\mathsf{pp}_{\mathsf{NYM}} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q) \xleftarrow{\$} \mathsf{BLGen}(1^\lambda)$.

$\mathsf{KeyGen}_{\mathsf{NYM}}$. *To generate key pair, do:*

1. *Choose* $\mathsf{usk} \xleftarrow{\$} \mathbb{G}_1$.
2. *Let* $\mathsf{upk} \leftarrow e(\mathsf{usk}, g_2)$.
3. *Return* $(\mathsf{usk}, \mathsf{upk})$.

$\mathsf{Gen}_{\mathsf{NYM}}$. *To generate a pseudonym, do:*

1. *Return* $e(\mathsf{usk}, \mathcal{H}(\mathsf{scope}))$, *where* $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_2^\times$ *denotes a random oracle.*

Theorem D.3. *If the DCo-DH Assumption holds, then the above construction is secure and unlinkable.*

Proof. Correctness follows by inspection, as $\mathcal{H}$ and the pairing e are both deterministic. The other two properties are proven on its own. First of all note that giving out upk does not matter: g_2 is a randomly chosen generator of $\mathbb{G}_2$ and can thus be seen as a special scope.

Collision-Resistance. Collision-resistance is proven by a sequence of games.

Game 0: The original collision-resistance game.

Game 1: Now abort, if the adversary was able to find a collision w.r.t. to the definition. Let $g^x = \mathcal{H}(\mathsf{scope}) \in \mathbb{G}_2^\times$, by definition of $\mathcal{H}$. That means that exactly one such x exists, even though potentially unknown, as $\mathbb{G}_2$ is cyclic. Further, $\mathsf{nym} = g_T^{xy}$ for some y, where $\mathsf{usk} = g_1^y$ for some (also potentially unknown y) by definition. Now, as the system works in prime-order groups, collisions cannot happen, i.e., if $g^{xy} = g^{xy'}$, then $y = y'$. This proves that the construction is perfectly collision-resistant.

Unlinkability. Now, unlinkability is proven by a sequence of games. Essentially, a distinguisher can be turned into a distinguisher of a Decisional Co-Diffie-Hellman challenge.

Game 0: The original unlinkability game.

Game 1: Simulate (lazy) the random-oracle $\mathcal{H}$ honestly, but keep the queries and the answers in an internal table T. This is only an internal change.

Game 2: Now abort, if $\mathsf{usk}_0 = \mathsf{usk}_1$. This can only happen with negligible probability due to the super-polynomial key-space.

Game 3: Abort, if the adversary makes a query h to $\mathcal{H}$ such that $\mathcal{H}(h) = g_2$. This only happens with negligible probability, i.e., $\frac{q_s}{|\mathbb{G}_2|-1}$ due to the random choice of the responses, where q_s is the number of queries made.

Game 4: No longer compute nym^* honestly, but with a fresh $\mathsf{usk}' \in \mathbb{G}_1$. This only changes the view of the adversary negligibly. To show this, consider the following reduction. First, the reduction $\mathcal{B}$ receives the challenge parameters $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q, g_1^x, g_2^y, g_2^z)$. It embeds $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T, q)$ in the public parameters. So far, this does not change the view of the adversary at all. However, abort if $\mathsf{usk}' = \mathsf{usk}_0$ or $\mathsf{usk}' = \mathsf{usk}_1$. This only happens with negligible probability due to the super-polynomial key-space, i.e., $2/|\mathbb{G}_1 - 1|$. The reduction $\mathcal{B}$ then starts simulating the random-oracle $\mathcal{H}$ in the following way. It is assumed that every query to the oracles for some scope, scope was already queried to the random oracle $\mathcal{H}$. Otherwise, the reduction $\mathcal{B}$ does that query. Namely, for each (fresh) query q to $\mathcal{H}$, it draws $r_q \xleftarrow{\$} \mathbb{Z}_q^*$, and sets the response to $g_2^{r_q}$, and stores (q, r_q) in an internal table T_q. So far, the distributions are equal. Draw $r \xleftarrow{\$} \mathbb{Z}_q^*$ and a random bit $d \xleftarrow{\$} \{0, 1\}$. Then, for every query scope to $\mathcal{O}_{1-d}$, it returns $e(g_1^x, g_2^{yr_q})$ (r_q taken from record (scope, r_q)) and for every query to $\mathcal{O}_d$, it returns $e(g_1^{rx}, g_2^{yr_q})$. So far, the distributions are still equal. Finally, return $e(g_1, g_2^{zr_q})$ for scope^*. If the adversary guesses d correctly, return 0. Otherwise, return 1. Clearly, if $z = xy$, the simulation is perfect. If, however, $z \neq xy$, then the exponent r_q is irrelevant, and the claim follows.

Clearly, nym^* is now completely decoupled from the bit b, from which unlinkability follows. □

Appendix E

Single Sign-On: Detailed Experimental Results

Here, the detailed measurements are presented.

E.1 The First Protocol

This section presents the detailed measurements for the first protocol. Each measurement is presented on its own page to increment readability.

E.1.1 Parameter Generation

Table E.1 contains the bare numbers for the parameter-generation for the first protocol. The corresponding boxplots are given in Figure E.1.

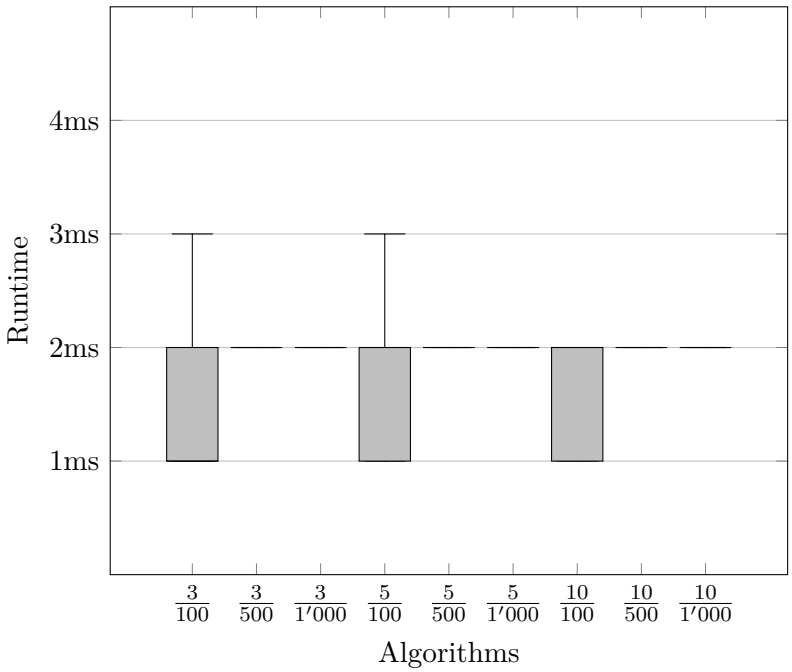

Figure E.1: Box-Plots of the parameter generation measurements in ms for the first protocol

Table E.1: Percentiles in milliseconds for parameter generation of the first protocol

$\lvert T \rvert$	3			5			10		
$\lvert S \rvert$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	1	1	2	1	1	2	1	1	2
25%:	1	2	2	1	2	2	1	2	2
Med.:	1	2	2	2	2	2	1	2	2
75%:	2	2	2	2	2	2	2	2	2
90%:	2	2	2	2	2	2	2	2	2
95%:	2	2	3	2	3	2	2	2	2
Max.:	3	2	3	3	3	3	2	3	2
Avg.:	2	2	2	2	2	2	2	2	2

This shows that parameter generation is constant, regardless how many ticket-granting servers or services are in the system.

E.1.2 Key-Generation Ticket-Granting Servers

Table E.2 contains the bare numbers for the key-generation measurements for the ticket-granting servers for the first protocol. The corresponding boxplots are given in Figure E.2. Note, the numbers are accumulated, i.e., the key-generation was not done in parallel.

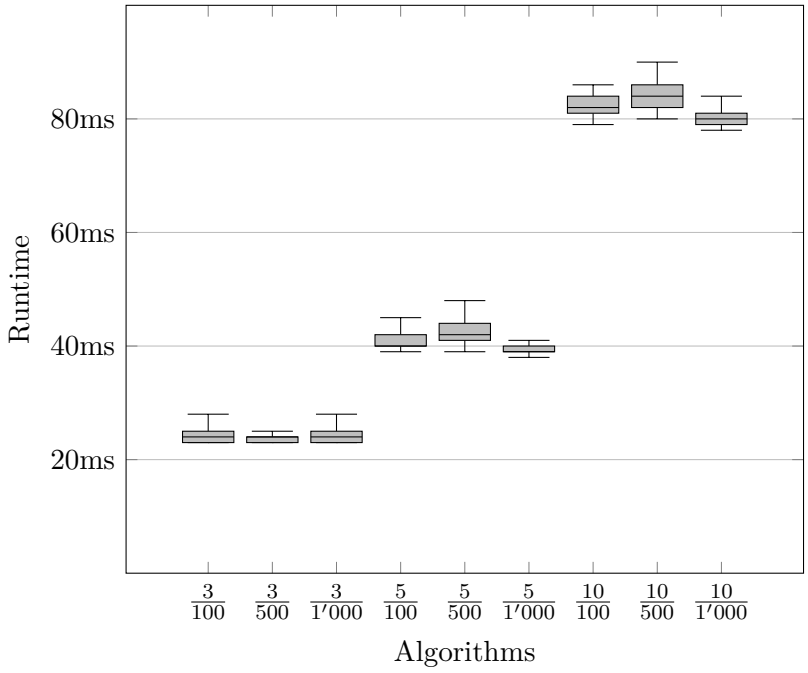

Figure E.2: Box-Plots of the key-generation measurements for the ticket-granting servers in ms for the first protocol

Table E.2: Percentiles for key-generation in milliseconds for the ticket-granting servers of the first protocol

$\|T\|$	3			5			10		
$\|S\|$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	23	23	23	39	39	38	79	80	78
25%:	23	23	23	40	41	39	81	82	79
Med.:	24	24	24	40	42	39	82	84	80
75%:	25	24	25	42	44	40	84	86	81
90%:	26	26	27	46	47	41	85	96	83
95%:	28	26	27	51	49	42	85	105	85
Max.:	31	32	30	55	55	43	86	117	87
Avg.:	34	24	24	42	43	40	82	87	81

This shows that key-generation for the ticket-granting servers only depend on the numbers of ticket-granting servers in the system.

E.1.3 Key-Generation Ticket-Granting Service

Table E.3 contains the bare numbers for the key-generation measurements for the services for the first protocol. The corresponding boxplots are given in Figure E.3. Note, the numbers are accumulated, i.e., the key-generation was not done in parallel.

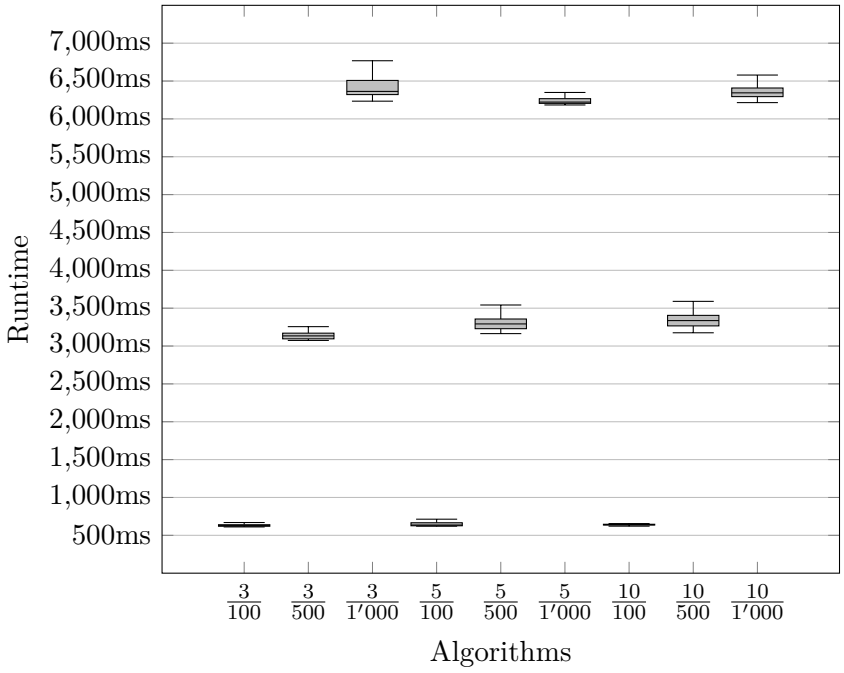

Figure E.3: Box-Plots of the key-generation measurements for the services in ms for the first protocol

Table E.3: Percentiles for key-generation in milliseconds for the services of the first protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	611	3′074	6′234	621	3′164	6′184	621	3′175	6′215
25%:	620	3′095	6′321	626	3′229	6′205	636	3′266	6′294
Med.:	628	3′134	6′364	636	3′293	6′223	640	3′336	6′344
75%:	640	3′170	6′508	666	3′357	6′267	646	3′405	6′408
90%:	663	3′299	6′814	703	3′444	6′309	651	3′481	6′467
95%:	674	3′461	6′877	771	3′477	6′349	653	3′555	6′555
Max.:	776	3′764	7′025	834	3′561	6′538	655	3′590	6′700
Avg.:	636	3′171	6′450	656	3′310	6′245	641	3′346	6′362

This shows that key-generation for the services only depend on the numbers of services in the system.

E.1.4 Registration User

Table E.4 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.4. Note, the numbers are accumulated, i.e., all computation done during the registration phase by the user is included.

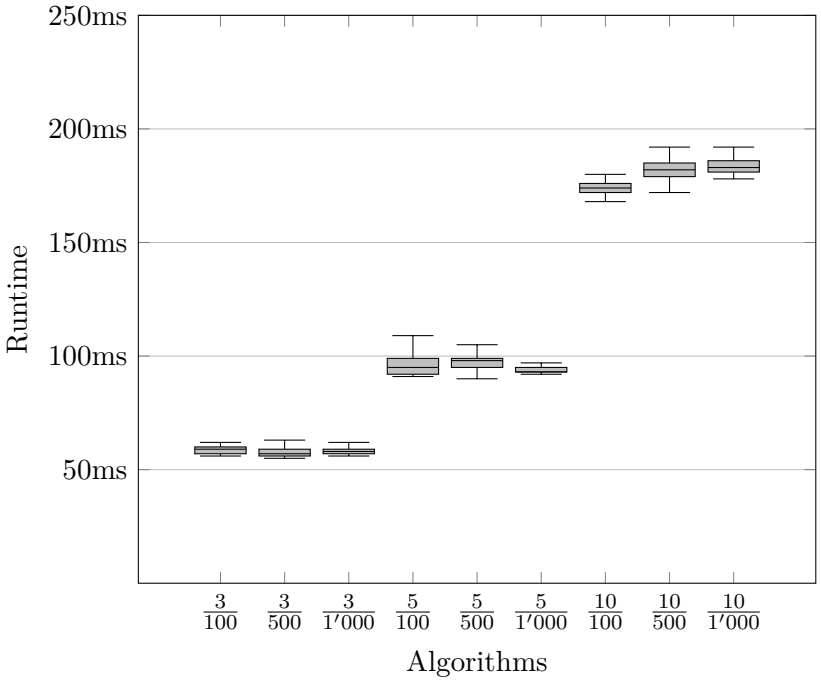

Figure E.4: Box-Plots of the user registration measurements in ms for the first protocol

Table E.4: Percentiles for registration in milliseconds for the user of the first protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	56	55	56	91	90	92	168	172	178
25%:	57	56	57	92	95	93	172	179	181
Med.:	59	57	58	95	98	93	174	182	183
75%:	60	59	59	99	99	95	176	185	186
90%:	62	60	62	104	104	96	178	196	189
95%:	67	63	63	108	106	97	178	199	191
Max.:	76	69	65	120	109	97	180	209	207
Avg.:	60	58	58	97	98	94	174	183	184

This shows that the registration phase of the user only depends on how many ticket-granting servers are in the system.

E.1.5 Registration Ticket-Granting Servers

Table E.5 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.5. Note, the numbers are accumulated, i.e., all computation done during the registration phase by the ticket-granting servers is included.

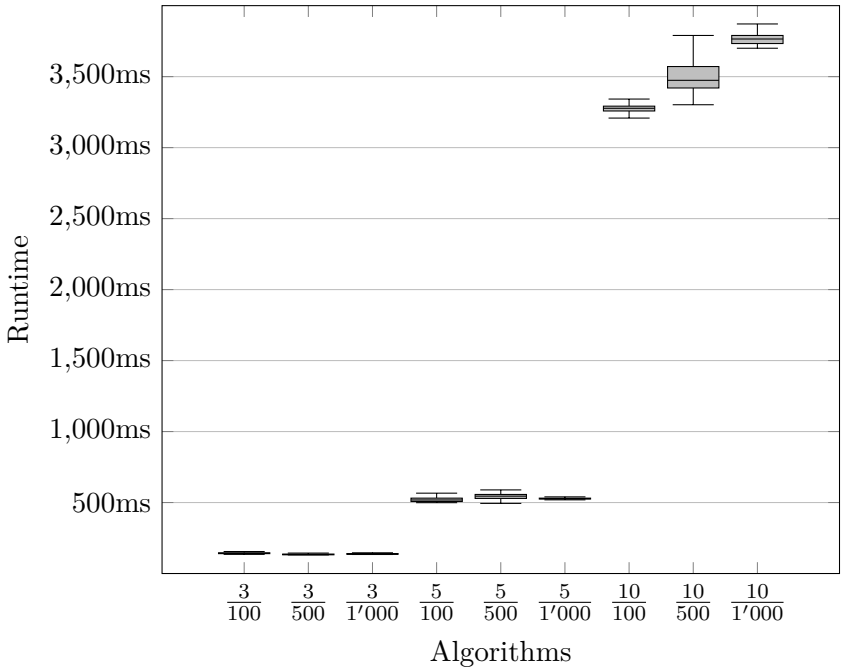

Figure E.5: Box-Plots of the ticket-granting server registration measurements in ms for the first protocol

Table E.5: Percentiles for registration in milliseconds for the ticket-granting servers of the first protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	135	131	134	449	494	519	3′168	3′302	3′700
25%:	140	132	136	507	529	525	3′258	3′420	3′733
Med.:	143	134	138	518	545	527	3′278	3′478	3′765
75%:	146	137	140	531	557	531	3′292	3′571	3′790
90%:	150	142	146	561	565	538	3′314	3′678	3′821
95%:	153	144	149	579	574	542	3′328	3′755	3′833
Max.:	160	150	158	693	589	555	3′342	4′037	3′876
Avg.:	144	135	140	526	542	529	3′276	3′509	3′767

This shows that the registration phase, as for the user, only depends on how many ticket-granting servers are in the system.

E.1.6 Token-Generation User

Table E.6 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.6. Note, the numbers are accumulated, i.e., all computation done during the token-generation phase by the user is included.

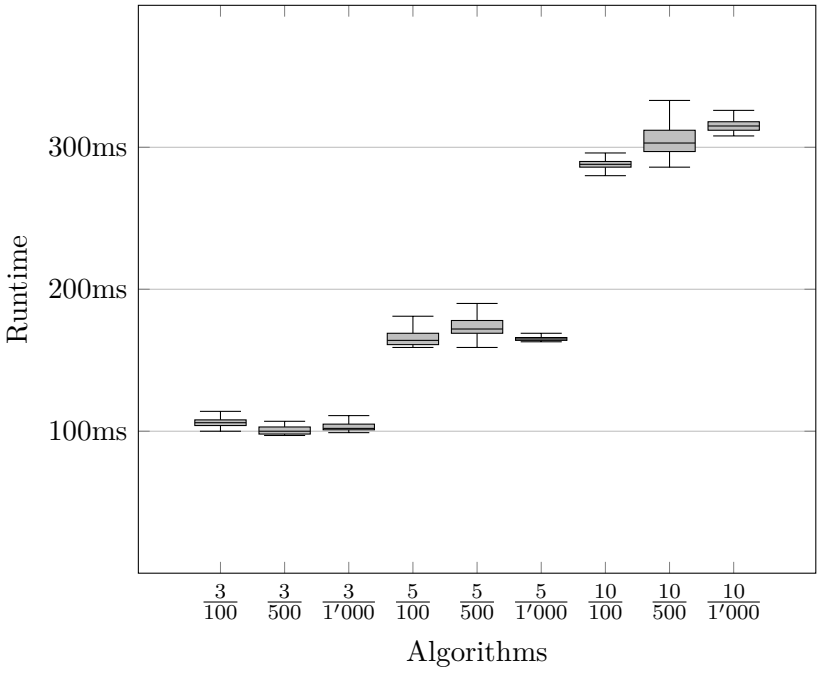

Figure E.6: Box-Plots of the user token-generation measurements in ms for the first protocol

Table E.6: Percentiles for token-generation in milliseconds for the user of the first protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	100	97	99	159	159	163	279	286	308
25%:	104	98	101	161	169	164	286	297	312
Med.:	106	100	102	164	172	165	288	303	315
75%:	108	103	105	169	178	166	290	312	318
90%:	112	105	107	183	182	168	292	329	321
95%:	116	107	109	195	186	170	293	342	322
Max.:	121	117	115	216	216	172	296	350	328
Avg.:	107	101	103	169	173	165	288	307	315

This shows that the token-generation phase mainly depends on how many ticket-granting servers are in the system.

E.1.7 Token-Generation Ticket-Granting Servers

Table E.7 contains the bare numbers for the registration measurements of the ticket-granting servers for the first protocol. The corresponding boxplots are given in Figure E.7. Note, the numbers are accumulated, i.e., all computation done during the token-generation phase by the ticket-granting servers is included.

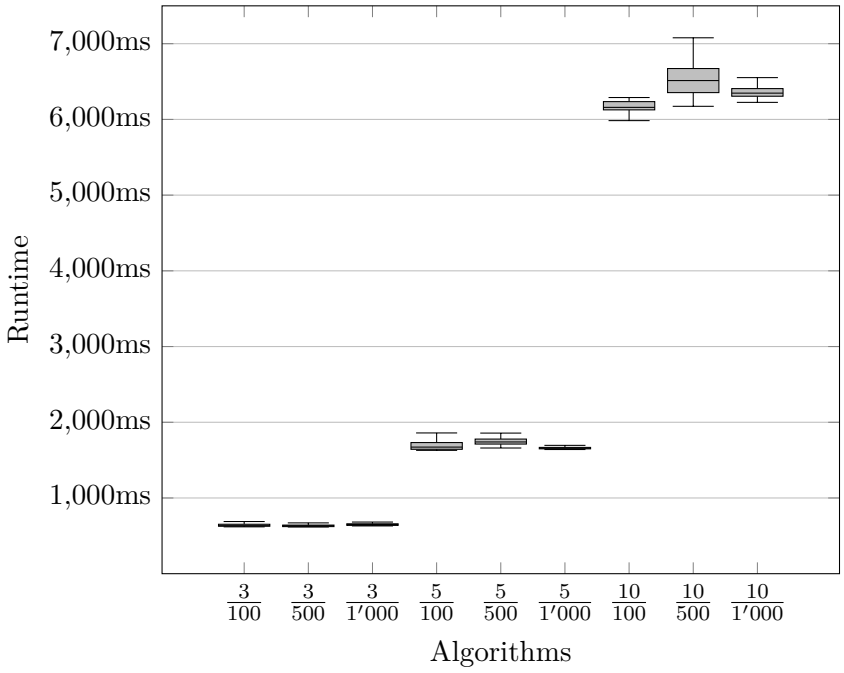

Figure E.7: Box-Plots of the ticket-granting servers token-generation measurements in ms for the first protocol

Table E.7: Percentiles for token-generation in milliseconds for the ticket-granting server of the first protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	619	617	631	1′629	1′660	1′640	5′984	6′173	6′225
25%:	627	622	638	1′643	1′712	1′649	6′126	6′354	6′306
Med.:	634	634	649	1′674	1′742	1′657	6′159	6′517	6′350
75%:	653	642	659	1′732	1′778	1′668	6′235	6′672	6′407
90%:	689	668	671	1′845	1′836	1′688	6′260	6′860	6′494
95%:	713	680	703	2′003	1′909	1′704	6′269	6′961	6′537
Max.:	779	743	785	2′176	1′994	1′732	6′289	7′519	6′646
Avg.:	647	639	654	1′716	1′758	1′663	6′168	6′550	6′369

This shows that the token-generation phase, as for the user, mainly depends on how many ticket-granting servers are in the system.

E.1.8 Communication User

Table E.8 contains the bare numbers for the communication measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.8.

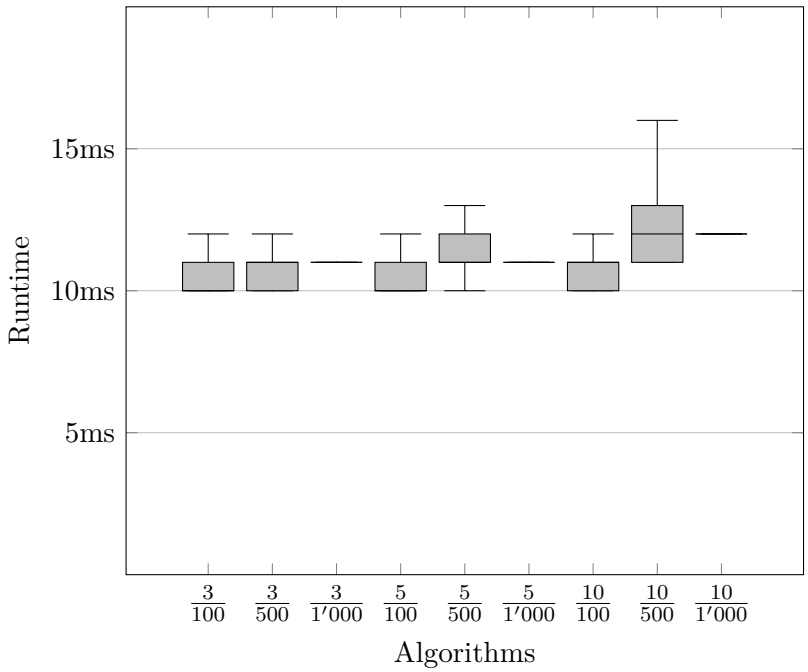

Figure E.8: Box-Plots of the user communication measurements in ms for the first protocol

Table E.8: Percentiles for communication in milliseconds for the user of the first protocol

$\lvert T \rvert$	3			5			10		
$\lvert S \rvert$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	10	10	11	10	10	11	10	11	12
25%:	10	10	11	10	11	11	10	11	12
Med.:	10	11	11	11	11	11	11	12	12
75%:	11	11	11	11	12	11	11	13	12
90%:	12	12	12	12	13	12	11	13	13
95%:	12	12	12	13	14	12	11	14	13
Max.:	12	13	15	14	16	12	12	16	14
Avg.:	11	11	11	11	12	11	11	12	12

This shows that the communication phase is essentially constant for the user.

E.1.9 Communication Service

Table E.9 contains the bare numbers for the communication measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.9.

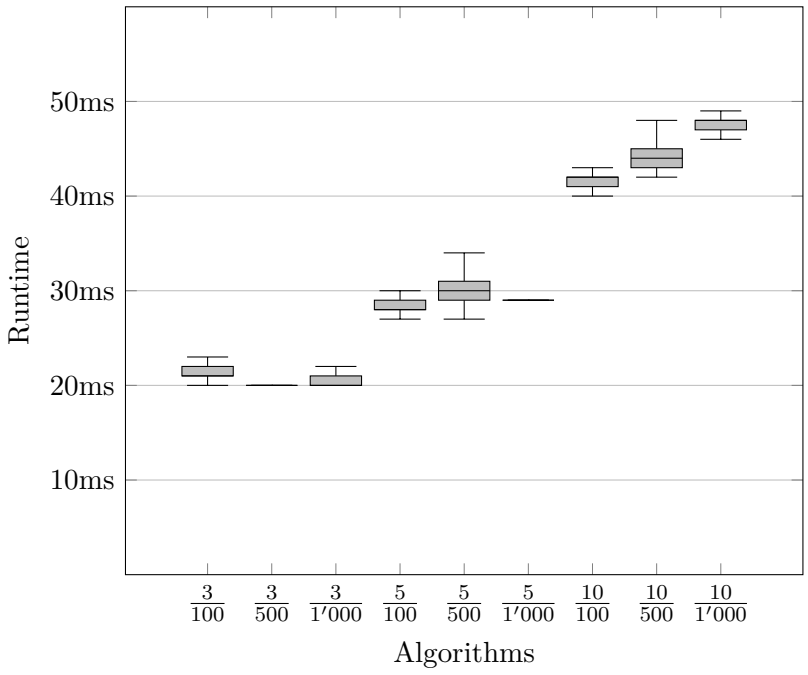

Figure E.9: Box-Plots of the service communication measurements in ms for the first protocol

Table E.9: Percentiles for communication in milliseconds for the service of the first protocol

$\|\mathsf{T}\|$	3			5			10		
$\|\mathsf{S}\|$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	20	19	20	27	27	28	40	42	46
25%:	21	20	20	28	29	29	41	43	47
Med.:	21	20	20	28	30	29	42	44	48
75%:	22	20	21	29	31	29	42	45	48
90%:	23	21	22	31	33	30	43	48	49
95%:	25	22	23	35	34	30	44	51	50
Max.:	29	24	24	43	37	31	44	59	52
Avg.:	22	20	21	29	30	29	42	45	48

This shows that the communication phase essentially only depends on how many ticket-granting servers are in the system.

E.2 The Second Protocol

This section presents the detailed measurements for the second, i.e., privacy-enhanced protocol. As for the first protocol, each measurement is presented on its own page to increment readability.

E.2.1 Parameter Generation

Table E.10 contains the bare numbers for the parameter-generation for the first protocol. The corresponding boxplots are given in Figure E.10.

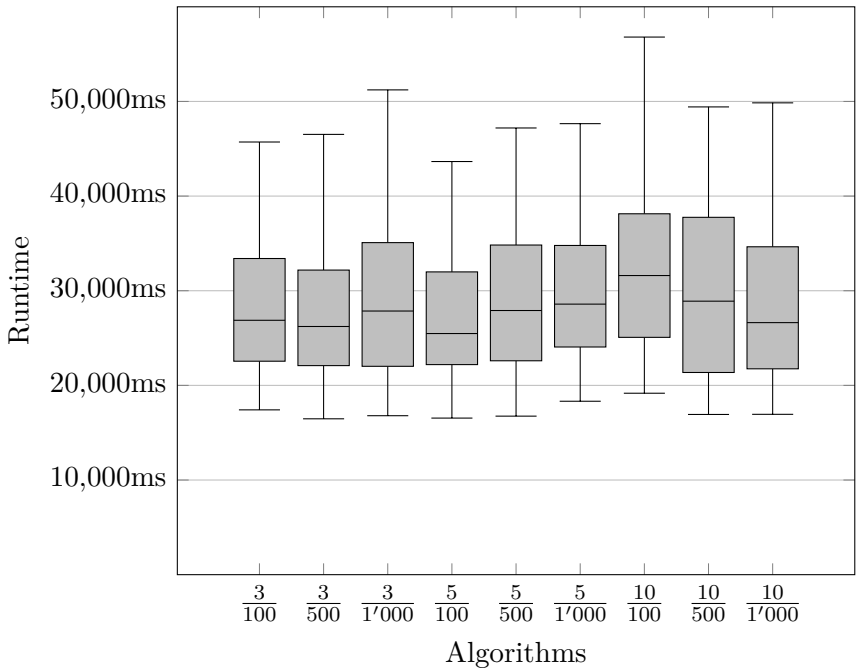

Figure E.10: Box-Plots of the parameter generation measurements in ms for the second protocol

Table E.10: Percentiles milliseconds for parameter generation of the second protocol

$\lvert\mathsf{T}\rvert$	3			5			10		
$\lvert\mathsf{S}\rvert$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	17′413	16′468	16′795	16′544	16′750	18′324	19′171	16′928	16′942
25%:	22′553	22′088	22′015	22′195	22′599	24′055	25′075	21′362	21′750
Med.:	26′958	26′234	28′131	25′552	28′001	28′593	31′683	29′085	26′718
75%:	33′405	32′183	35′083	31′990	34′829	34′784	38′131	37′757	34′644
90%:	39′660	39′001	41′352	38′576	40′678	40′825	45′648	43′033	38′967
95%:	42′337	42′998	47′289	42′078	43′017	44′946	50′762	45′750	43′345
Max.:	45′714	51′123	51′221	43′651	47′197	47′653	56′802	49′422	49′851
Avg.:	28′705	28′066	29′630	27′433	28′965	29′854	32′719	30′057	28′727

This shows that parameter generation is constant, regardless how many ticket-granting servers or services are in the system. The lion's share, however, is finding safe prime for the encryption scheme.

E.2.2 Key-Generation Ticket-Granting Servers

Table E.11 contains the bare numbers for the key-generation measurements for the ticket-granting servers for the first protocol. The corresponding boxplots are given in Figure E.11. Note, the numbers are accumulated, i.e., the key-generation was not done in parallel.

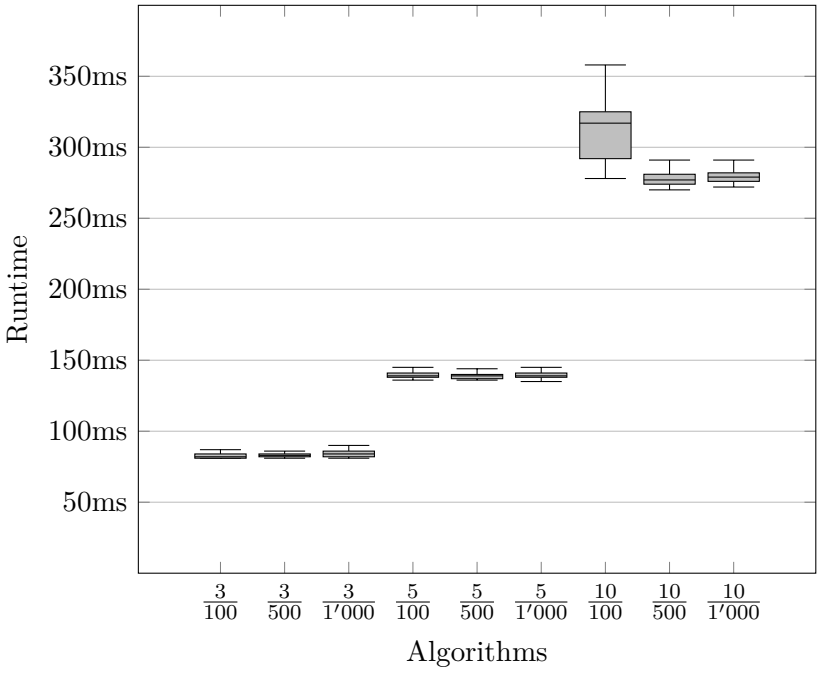

Figure E.11: Box-Plots of the key-generation measurements for the ticket-granting servers in ms for the second protocol

Table E.11: Percentiles for key-generation in milliseconds for the ticket-granting servers of the second protocol

$\|T\|$	3			5			10		
$\|S\|$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	81	81	81	136	136	135	278	270	272
25%:	81	82	82	138	137	138	292	274	276
Med.:	82	83	84	139	139	139	318	277	279
75%:	84	84	86	141	140	141	325	281	282
90%:	85	85	89	142	144	143	334	291	289
95%:	86	85	90	144	146	145	342	296	291
Max.:	87	88	97	150	153	150	358	316	308
Avg.:	83	83	85	140	139	140	313	280	280

This shows that key-generation for the ticket-granting servers only depend on the numbers of ticket-granting servers in the system.

E.2.3 Key-Generation Ticket-Granting Service

Table E.12 contains the bare numbers for the key-generation measurements for the services for the first protocol. The corresponding boxplots are given in Figure E.12. Note, the numbers are accumulated, i.e., the key-generation was not done in parallel.

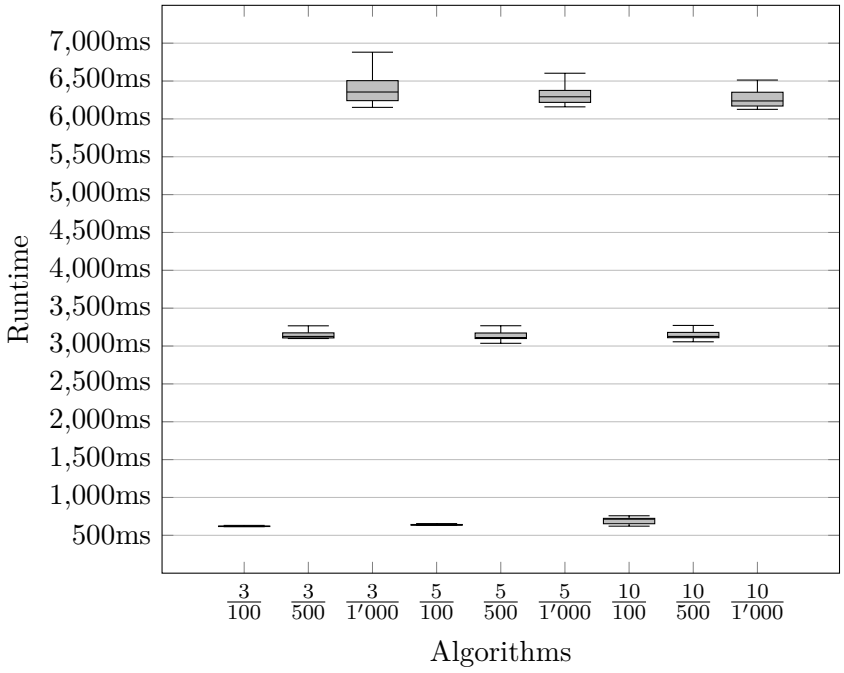

Figure E.12: Box-Plots of the key-generation measurements for the services in ms for the second protocol

Table E.12: Percentiles for key-generation in milliseconds for the services of the second protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	614	3′099	6′152	615	3′036	6′159	621	3′056	6′126
25%:	616	3′107	6′241	634	3′099	6′218	653	3′110	6′170
Med.:	619	3′127	6′360	638	3′113	6′292	713	3′127	6′237
75%:	622	3′174	6′505	643	3′172	6′376	724	3′180	6′352
90%:	634	3′212	6′657	650	3′230	6′453	738	3′303	6′427
95%:	642	3′242	6′775	653	3′240	6′529	750	3′374	6′513
Max.:	651	3′307	7′169	684	3′299	6′681	758	3′566	6′766
Avg.:	622	3′146	6′403	639	3′140	6′310	697	3′164	6′271

This shows that key-generation for the services only depend on the numbers of services in the system.

E.2.4 Registration User

Table E.13 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.13. Note, the numbers are accumulated, i.e., all computation done during the registration phase by the user is included.

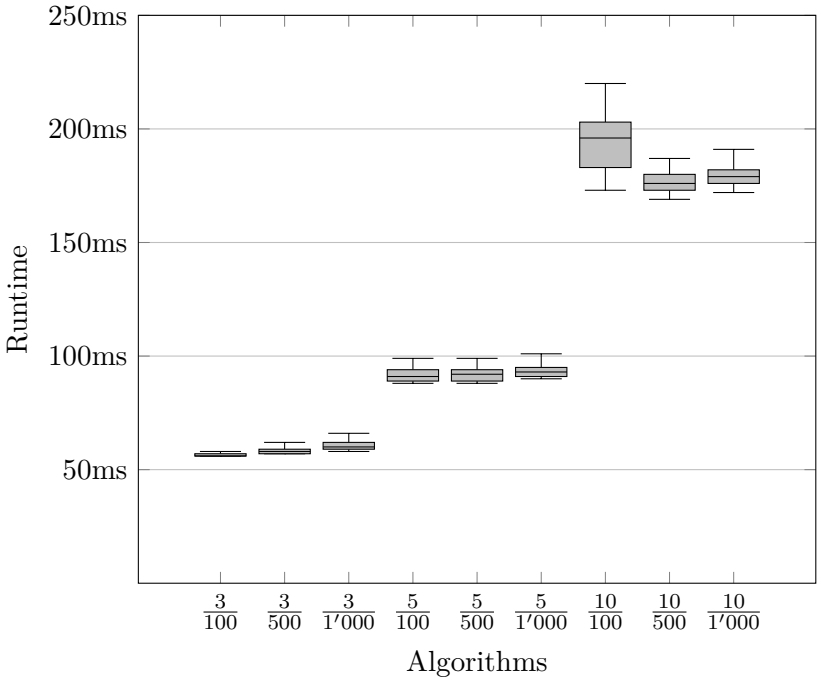

Figure E.13: Box-Plots of the user registration measurements in ms for the second protocol

Table E.13: Percentiles for registration in milliseconds for the user of the second protocol

$\|T\|$	3			5			10		
$\|S\|$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	56	57	58	88	88	90	173	169	172
25%:	56	57	59	89	89	91	183	173	176
Med.:	56	58	60	91	92	93	196	176	179
75%:	57	59	62	94	94	95	203	180	182
90%:	59	61	64	96	95	97	211	186	185
95%:	60	62	66	97	97	99	214	195	188
Max.:	64	64	73	99	99	101	220	244	195
Avg.:	57	58	61	92	92	94	195	179	180

This shows that the registration phase of the user only depends on how many ticket-granting servers are in the system.

E.2.5 Registration Ticket-Granting Servers

Table E.14 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.14. Note, the numbers are accumulated, i.e., all computation done during the registration phase by the ticket-granting servers is included.

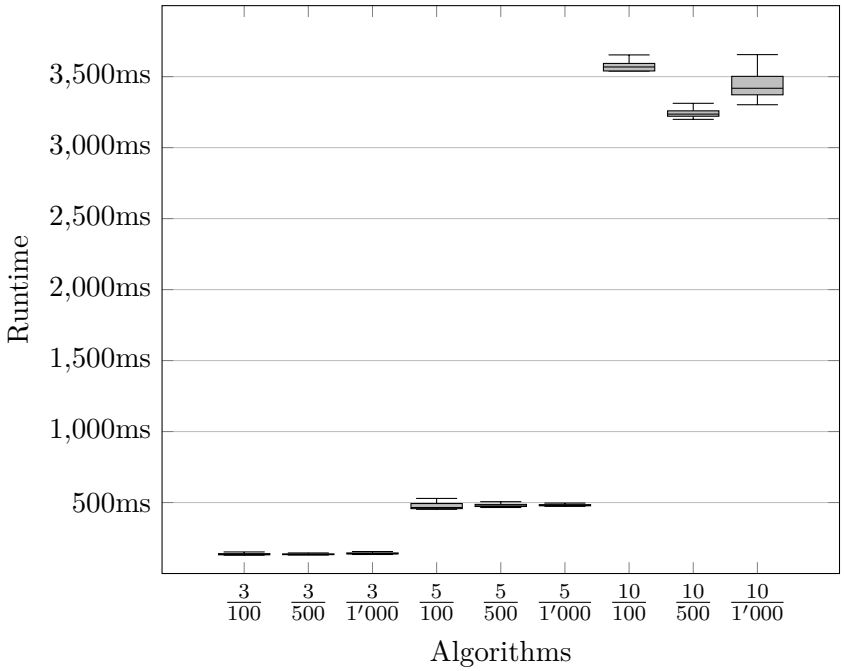

Figure E.14: Box-Plots of the ticket-granting server registration measurements in ms for the second protocol

Table E.14: Percentiles for registration in milliseconds for the ticket-granting servers of the second protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	129	130	134	452	465	473	3′157	3′199	3′302
25%:	132	133	138	457	472	477	3′540	3′221	3′372
Med.:	135	135	140	465	476	481	3′568	3′236	3′420
75%:	140	138	145	493	487	485	3′593	3′259	3′502
90%:	148	145	155	506	498	491	3′609	3′274	3′588
95%:	154	149	162	508	502	496	3′637	3′281	3′655
Max.:	172	155	172	529	506	502	3′653	3′339	3′762
Avg.:	138	137	143	474	480	482	3′495	3′242	3′449

This shows that the registration phase, as for the user, only depends on how many ticket-granting servers are in the system.

E.2.6 Token-Generation User

Table E.15 contains the bare numbers for the registration measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.15. Note, the numbers are accumulated, i.e., all computation done during the token-generation phase by the user is included.

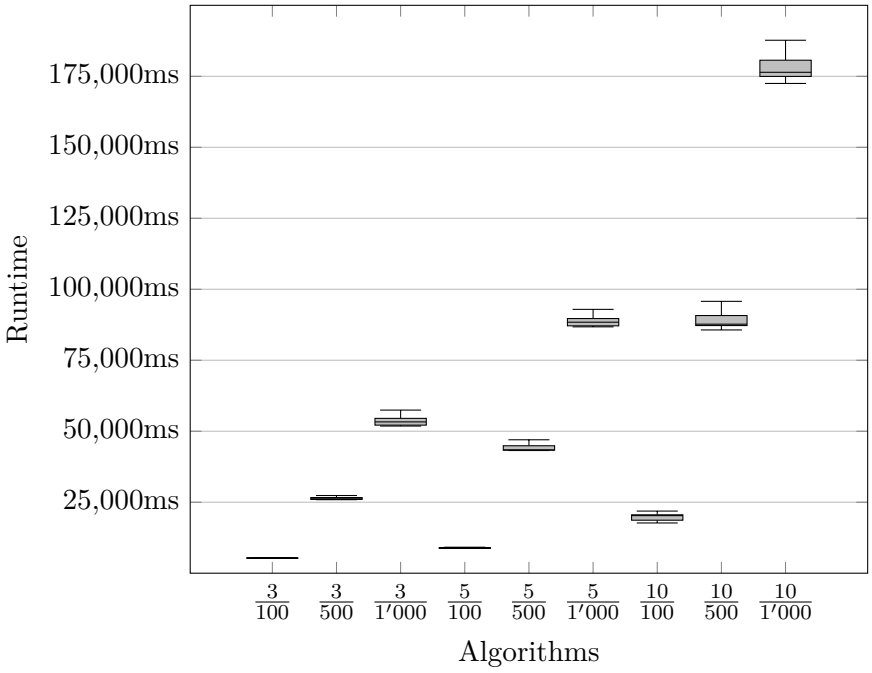

Figure E.15: Box-Plots of the user token-generation measurements in ms for the second protocol

Table E.15: Percentiles for token-generation in milliseconds for the user of the second protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	5′284	25′890	51′826	8′745	43′200	86′762	17′693	85′669	172′481
25%:	5′301	25′982	52′144	8′772	43′318	87′117	18′661	87′211	174′953
Med.:	5′319	26′210	53′392	8′802	43′437	88′400	20′279	87′714	176′417
75%:	5′350	26′649	54′535	8′953	44′885	89′686	20′565	90′742	180′667
90%:	5′481	26′913	55′686	9′083	46′225	91′543	20′987	93′881	184′230
95%:	5′516	27′207	56′173	9′151	46′540	92′366	21′248	94′999	185′263
Max.:	5′566	28′088	5′7443	9′373	47′948	94′149	21′877	98′233	189′553
Avg.:	5′349	26′338	53′578	8′879	44′162	88′709	19′784	89′023	177′998

In contrast to the first protocol, the token-generation now depends on the number of ticket-granting servers and services in the system.

E.2.7 Token-Generation Ticket-Granting Servers

Table E.16 contains the bare numbers for the registration measurements of the ticket-granting servers for the first protocol. The corresponding boxplots are given in Figure E.16. Note, the numbers are accumulated, i.e., all computation done during the token-generation phase by the ticket-granting servers is included.

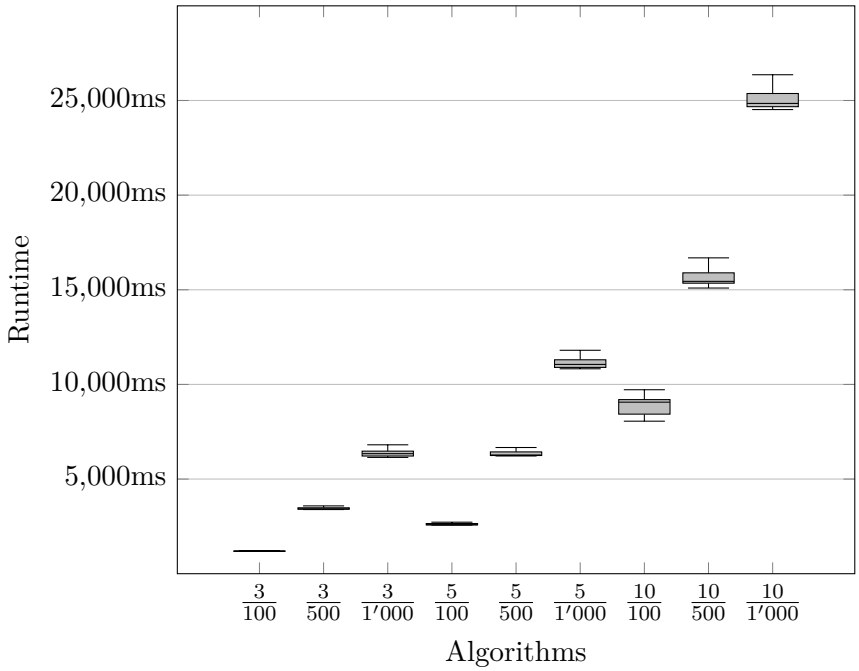

Figure E.16: Box-Plots of the ticket-granting servers token-generation measurements in ms for the second protocol

Table E.16: Percentiles for token-generation in milliseconds for the ticket-granting server of the second protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	1′178	3′389	6′144	2′567	6′221	10′826	8′058	15′092	24′520
25%:	1′183	3′407	6′217	2′576	6′244	10′899	8′430	15′348	24′673
Med.:	1′187	3′423	6′342	2′592	6′272	11′063	9′068	15′445	24′851
75%:	1′199	3′483	6′475	2′652	6′435	11′302	9′198	15′895	25′369
90%:	1′224	3′533	6′654	2′681	6′547	11′484	9′333	16′436	25′739
95%:	1′247	3′561	6′780	2′707	6′599	11′568	9′555	16′857	26′085
Max.:	1′304	3′627	7′485	2′780	6′726	11′932	9′719	17′265	26′541
Avg.:	1′197	3′450	6′396	2′616	6′342	11′124	8′894	15′677	25′054

In contrast to the first protocol, the token-generation now depends on the number of ticket-granting servers and services in the system.

E.2.8 Communication User

Table E.17 contains the bare numbers for the communication measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.17.

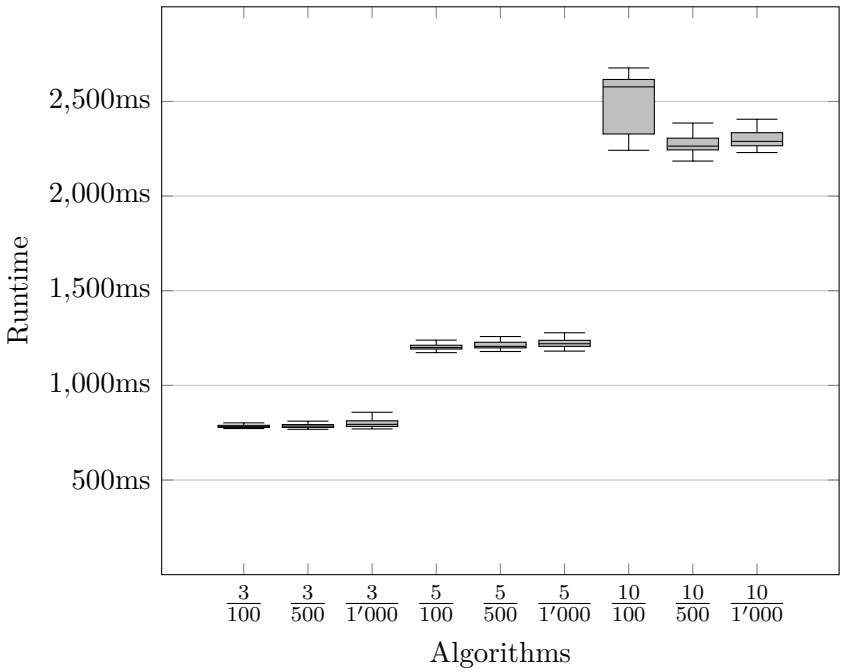

Figure E.17: Box-Plots of the user communication measurements in ms for the second protocol

Table E.17: Percentiles for communication in milliseconds for the user of the second protocol

|T|	3			5			10		
|S|	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	772	768	770	1′173	1′179	1′181	2′242	2′185	2′230
25%:	778	778	783	1′192	1′198	1′206	2′328	2′244	2′266
Med.:	782	785	794	1′200	1′206	1′221	2′580	2′264	2′289
75%:	789	793	813	1′212	1′228	1′238	2′616	2′306	2′335
90%:	801	802	826	1′225	1′246	1′259	2′651	2′415	2′378
95%:	812	806	832	1′231	1′258	1′287	2′662	2′488	2′396
Max.:	827	811	870	1′271	1′345	1′319	2′677	2′603	2′462
Avg.:	786	786	799	1′204	1′215	1′226	2′505	2′294	2′305

This shows that the communication phase, in contrast to the first protocol, now also depends on how many ticket-granting servers are in the system.

E.2.9 Communication Service

Table E.18 contains the bare numbers for the communication measurements of the user for the first protocol. The corresponding boxplots are given in Figure E.18.

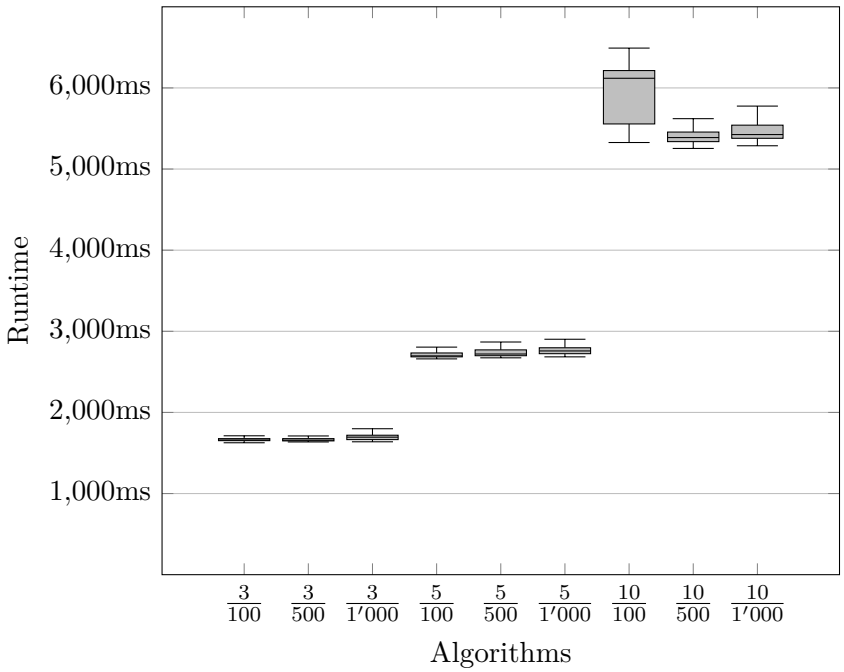

Figure E.18: Box-Plots of the service communication measurements in ms for the second protocol

Table E.18: Percentiles for communication in milliseconds for the service of the second protocol

$\lvert T \rvert$	3			5			10		
$\lvert S \rvert$	100	500	1′000	100	500	1′000	100	500	1′000
Min.:	1′626	1′636	1′638	2′660	2′673	2′685	5′327	5′254	5′287
25%:	1′652	1′649	1′665	2′685	2′699	2′724	5′556	5′338	5′379
Med.:	1′660	1′658	1′695	2′697	2′720	2′759	6′122	5′388	5′426
75%:	1′677	1′677	1′719	2′733	2′771	2′797	6′214	5′456	5′541
90%:	1′701	1′701	1′771	2′784	2′816	2′819	6′266	5′621	5′628
95%:	1′712	1′722	1′801	2′805	2′858	2′898	6′300	5′764	5′716
Max.:	1′723	1′738	1′863	2′907	3′008	2′961	6′491	6′093	5′902
Avg.:	1′667	1′667	1′702	2′716	2′744	2′767	5′957	5′431	5′466

This shows that the communication phase, in contrast to the first protocol, now also depends on how many ticket-granting servers are in the system.

Appendix F

Proof of Theorem 4.4

This section is devoted to prove Theorem 4.4, i.e., showing that the modified Bellare-Rogaway encryption as described in Section 4.1.3 achieves the notion of RECV-SIM security in the random-oracle model.

Proof. The proof is related to the one given by Nielsen [Nie02] for his construction. The proof also reduces the security of our construction to the one-wayness of the underlying trapdoor permutation. For the sake of readability, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ is described first. Afterwards, is proven, via a sequence of games, that the scheme with the given simulator is RECV-SIM-secure, as required by definition. The simulator processes its tasks as follows.

Key Generation. On input of $(\mathsf{PUBLICKEY}, 1^\lambda)$, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ generates $(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$. It stores $\mathsf{sk}_{\mathsf{ENC}} = f^{-1}$, and returns (f, Σ) as $\mathsf{pk}_{\mathsf{ENC}}$.

Encryption. On input of $(\mathsf{ENCRYPT}, k, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $r \xleftarrow{\$} \Sigma$. It also saves the randomness r for further usage. Namely, if the same r was drawn before, the simulator aborts. Let $l \leftarrow |\mathsf{ec}(1^k)|/\lambda$.[1] Set $c_{(1)} \leftarrow f(r)$, $c^i_{(2)} \xleftarrow{\$} \{0,1\}^\lambda$, for all $0 < i \leq l$, and $c_{(3)} \xleftarrow{\$} \{0,1\}^\lambda$. $\mathsf{SIM}_{\mathsf{NCE}}$ returns $c \leftarrow (c_{(1)}, (c^1_{(2)}, c^2_{(2)}, \ldots, c^2_{(l)}), c_{(3)})$.

RO Queries. With the input of $(\mathsf{ROQUERY}, q)$ both random oracles $\mathcal{G}$ and $\mathcal{K}$ are addressed, i.e., it is assumed that q is prefixed accordingly (e.g., $q = (\mathcal{G}, q')$). If the input does not satisfy this format, $\mathsf{SIM}_{\mathsf{NCE}}$ ignores the inputs (in the real world, the same behavior is enforced by definition). $\mathcal{L}_\mathcal{G}$ and $\mathcal{L}_\mathcal{K}$ is used as a short-hand notation to refer to the lists of queries and answers to $\mathcal{G}$ and $\mathcal{K}$, respectively. If for any of the random-oracle queries the simulator draws a response twice, it aborts.

Decryption. On input $(\mathsf{DECRYPT}, (c_{(1)}, (c^1_{(2)}, c^2_{(2)}, \ldots, c^k_{(2)}), c_{(3)}), \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ checks if a preimage (r', k, m', ℓ) of $c_{(3)}$ for $\mathcal{K}$ and images of (i, r') for $\mathcal{G}$ and all $1 < i \leq k$ have been defined. Let $(m'_1, m'_2, \ldots, m'_{k'}) \leftarrow \mathsf{ec}(m')$. If $k \neq k'$, the simulator returns $\perp$. It then checks whether $c^i_{(2)} = \mathcal{G}(i, r') \oplus m'_i$ for all $1 < i \leq k$, and $c_{(1)} = f(r')$ holds. If so $\mathsf{SIM}_{\mathsf{NCE}}$ returns m'. Otherwise, the simulator makes the missing oracle calls. If these make the ciphertext valid, the simulator aborts, otherwise it returns $\perp$.

[1] Note, this is always an integer.

Key Leakage. On input of $(\mathsf{KEYLEAK}, \mathcal{Q})$, $\mathsf{SIM}_{\mathsf{NCE}}$ must program the random oracles to ensure consistent decryption of ciphertexts as the adversary will receive the secret key $\mathsf{sk}_{\mathsf{ENC}}$. That is, for all $c_j = (c_{1,j}, (c^1_{(2,j)}, c^2_{(2,j)}, \ldots, c^{k_j}_{(2,j)}), c_{(3,j)})$, and $m_j \in \mathcal{Q}$, it programs $\mathcal{G}$ and $\mathcal{K}$ as follows: Let $(m_{1,j}, \ldots m_{k_j,j}) \leftarrow \mathsf{ec}(m_j)$. $\mathsf{SIM}_{\mathsf{NCE}}$ adds $((i, (f^{-1}(c_{(1,j)}))), m_{i,j} \oplus c^i_{(2,j)})$ for $0 < i \leq k_j$ to $\mathcal{L}_{\mathcal{G}}$ and adds $((f^{-1}(c_{(1,j)}), k_j, m_j, \ell_j), c_{(3,j)})$ to $\mathcal{L}_{\mathcal{K}}$. Finally, $\mathsf{SIM}_{\mathsf{NCE}}$ returns the secret key $\mathsf{sk}_{\mathsf{ENC}}$, i.e., f^{-1}. If the programming fails at some point, i.e., a preimage already exists, the simulator aborts.

Now is shown that the simulator is such that the adversary will output 1 in both experiments with essentially the same probability. By $\mathcal{P}_i$ the probability that $\mathcal{A}$ outputs 1 in Game i is denoted.

Game 0: In the first game, the real protocol is run with the adversary, i.e., RECV-SIM-real. Hence, it holds that:

$$\Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM\text{-}real}}_{\mathsf{ENC},\mathcal{A}}(1^\lambda) = 1] = \mathcal{P}_0$$

Game 1: Next, each query (m_i, ℓ_i) to $\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}}$ and the corresponding answer c_i is stored in a list $\mathcal{L}_{\mathsf{Enc}_{\mathsf{ENC}}}$. In particular, it is encrypted honestly, but $((m_i, \ell_i), c_i)$ is stored in a list $\mathcal{L}_{\mathsf{Enc}_{\mathsf{ENC}}}$. Whenever the adversary queries the decryption oracle with (c_i, ℓ_i), m_i is returned, if an entry $((m_i, \ell_i), c_{(i)}) \in \mathcal{L}_{\mathsf{Enc}_{\mathsf{ENC}}}$ for some m_i exists. In other words, such ciphertexts are not decrypted anymore. This is only an internal change, due to the perfect correctness of the scheme it holds that:

$$\left|\mathcal{P}_0 - \mathcal{P}_1\right| = 0$$

Game 2: Now, start gradually building the simulator $\mathsf{SIM}_{\mathsf{NCE}}$. In the first step, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ receives control of the random oracles $\mathcal{G}$ and $\mathcal{K}$. Abort, if the simulator draws a response twice. This only happens with probability $\frac{q_h^2}{2^\lambda}$ due to the birthday paradox, where q_h is the number of random oracle queries made. Thus, it holds that:

$$\left|\mathcal{P}_1 - \mathcal{P}_2\right| \leq \frac{q_h^2}{2^\lambda}$$

Game 3: Next, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ generates the key pair. In particular, on input of $(\mathsf{PUBLICKEY}, 1^\lambda)$, it generates $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$, and returns $(\mathsf{pk}_{\mathsf{ENC}}, \mathsf{sk}_{\mathsf{ENC}})$. This step is purely conceptional, so it holds that:

$$\left|\mathcal{P}_2 - \mathcal{P}_3\right| = 0$$

Game 4: Now, the encryption oracle $\mathcal{O}^{\mathsf{Enc}_{\mathsf{ENC}}}$ is simulated by $\mathsf{SIM}_{\mathsf{NCE}}$, i.e., on input of $(\mathsf{ENCRYPT}, m, \ell)$ (the simulator at this point still receives the full message m), the simulator draws $r \xleftarrow{\$} \Sigma$, calculates $c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, m, \ell; r)$, and returns (r, c). Note, Σ is exponential in size. However, the simulator aborts, if it draws a randomness r_i a twice. This only happens with probability at most $\frac{q^2}{|\Sigma|}$ due to the birthday paradox, where q is the number of queries made to $\mathsf{Enc}_{\mathsf{ENC}}$. Hence, it holds that:

$$\left|\mathcal{P}_3 - \mathcal{P}_4\right| \leq \frac{q^2}{|\Sigma|}$$

Game 5: Next, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ takes over the decryption of cipher-texts. On input of $(\mathsf{DECRYPT}, c, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ runs the decryption algorithms and returns the results. This is only a conceptual change, so it holds that:

$$\left|\mathcal{P}_4 - \mathcal{P}_5\right| = 0$$

Game 6: The simulator $\mathsf{SIM}_{\mathsf{NCE}}$ now no longer returns $\mathsf{sk}_{\mathsf{ENC}}$ directly after creation, but only when it is queried with $(\mathsf{KEYLEAK}, \cdot)$, which is sent once the adversary is finished with its query phase. As $\mathsf{sk}_{\mathsf{ENC}}$ is not required in the meantime, this is a conceptional change only, thus the following is true:

$$\left|\mathcal{P}_5 - \mathcal{P}_6\right| = 0$$

Game 7: The simulator now always returns $\perp$ whenever the adversary queries the decryption oracle with a ciphertext c for which not all random oracle calls have been made. In particular, the simulator checks whether a preimage (r', k, m, ℓ) of $c_{(3)}$ for $\mathcal{K}$, and images of (i, r') for $\mathcal{G}$ and all $1 < i \leq k$ have been defined and whether $c^i_{(2)} = \mathcal{G}(i, r') \oplus m_i$ for all $1 < i \leq k$ and $c_{(1)} = f(r')$ holds, while $k = \frac{|\mathsf{ec}(1^k)|}{\lambda}$ must hold. If any of these checks fail, the simulator outputs $\perp$. Note, the preimages are unique if they exist. Also, the simulator makes the missing random oracle calls. If these turn the ciphertext into a valid one, the simulator aborts. The adversary will only notice a difference to the Game 6 if the latter happens, which is with probability at most $\frac{q_h q_d}{2^\lambda}$, where q_h is the number of random oracle queries made and q_d denotes the number of calls to the decryption oracle. Hence, it holds that:

$$\left|\mathcal{P}_6 - \mathcal{P}_7\right| \leq \tfrac{q_h q_d}{2^\lambda}$$

Game 8: Now gradually change the simulator's input; instead of giving $\mathsf{SIM}_{\mathsf{NCE}}$ the message m before it has to come up with an encryption c, the challenger provides m after the adversary is done with its query phase, i.e., when the simulator receives $(\mathsf{KEYLEAK}, \mathcal{Q})$. In particular, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ only receives $(\mathsf{ENCRYPT}, |m|, \ell)$ instead of $(\mathsf{ENCRYPT}, m, \ell)$. Eventually, $(\mathsf{KEYLEAK}, \mathcal{Q})$ (containing the messages the cipher-texts should contain) is sent to $\mathsf{SIM}_{\mathsf{NCE}}$, which allows $\mathsf{SIM}_{\mathsf{NCE}}$ to program the random oracles accordingly. It is now proven that this only affects the view of the adversary negligibly by a series of hybrids. Let q be an upper bound of queries to the encryption oracle.

Hybrid 8.j: Up to encryption query j, where $0 \leq j \leq q$, the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ is given $(\mathsf{ENCRYPT}, |m|, \ell)$ instead of $(\mathsf{ENCRYPT}, m, \ell)$. So, Game 8.0 is identical to Game 7. On input $(\mathsf{ENCRYPT}, m, \ell)$, the simulator behaves as before. Now is described how the simulator $\mathsf{SIM}_{\mathsf{NCE}}$ proceeds on inputs $(\mathsf{ENCRYPT}, |m|, \ell)$. It draws $r \xleftarrow{\$} \Sigma$ as in $\mathsf{Enc}_{\mathsf{ENC}}$ and sets $c_{(1)} \leftarrow f(r)$. Let $l \leftarrow \frac{|\mathsf{ec}(1^{|m|})|}{\lambda}$. For each $0 < i \leq l$, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $c^i_{(2)} \xleftarrow{\$} \{0,1\}^\lambda$. Finally, it draws $c_{(3)} \xleftarrow{\$} \{0,1\}^\lambda$ and returns $c \leftarrow (c_{(1)}, c_{(2)}, c_{(3)})$. Note, the distributions of c are exactly the same as honest encryptions for the same message length $|m|$.

Eventually, $\mathsf{SIM}_{\mathsf{NCE}}$ is queried $(\mathsf{KEYLEAK}, \mathcal{Q})$ and thus receives the message m corresponding to each $(\mathsf{ENCRYPT}, |m|, \ell)$ call made earlier. To achieve consistent decryption, the simulator now programs the random oracles $\mathcal{K}$ and $\mathcal{G}$, such that a decryption of each generated c (with

the correct label ℓ) returns the corresponding m. Let $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{ec}(m)$. $\mathsf{SIM}_{\mathsf{NCE}}$ adds $((i, r), m_i \oplus c^i_{(2)})$ for each block m_i, $0 < i \leq k$ in m to $\mathcal{L}_{\mathcal{G}}$. Additionally, it adds $((r, k, m, \ell), c_{(3)})$ to $\mathcal{L}_{\mathcal{K}}$. This programming may fail if one of these preimages already exists in $\mathcal{L}_{\mathcal{G}}$ or $\mathcal{L}_{\mathcal{K}}$. This is only possible if the adversary had already queried (r, k, m, ℓ) to $\mathcal{K}$ or (i, r) for some i, $0 < i \leq k$ to $\mathcal{G}$, as $\mathsf{SIM}_{\mathsf{NCE}}$ always draws fresh random coins.

Reduction: Assuming that the adversary can distinguish between Hybrid $8.j$ and Hybrid $8.j + 1$, One can turn the adversary $\mathcal{A}$ into an adversary $\mathcal{B}$ which inverts a given element c, i.e., outputs $f^{-1}(c)$ with non-negligible probability. To do so, adversary $\mathcal{B}$ receives c and the corresponding parameters Σ and f from the trapdoor game. It embeds the challenge c for $\mathcal{A}$ as follows. It draws a random index $j \xleftarrow{\$} \{1, 2, \ldots, l\}$ and then on the jth query to the encryption oracle, $c_{(1)}$ is not calculated from a honestly drawn r, but is set to the provided challenge c, i.e., $c_{(1)} \leftarrow c$. Every other query is processed as in the prior game. In particular, one can extract $r = f^{-1}(c_{(1)})$ from $\mathcal{L}_{\mathcal{G}}$ or $\mathcal{L}_{\mathcal{K}}$ resp., and can therefore simulate the decryption oracle, if queried with a correctly computed ciphertext. Note that other ciphertexts are already excluded. Hence, the embedding does not change the view of the adversary. Assuming that one cannot program the random oracles accordingly, i.e., the adversary notices a difference to the prior game, then the adversary must have made a query (r', k', m', ℓ') to the random oracle $\mathcal{K}$, such that $c = c_{(1)} = f(r')$, or a query (i, r') for some i, $0 < i \leq k$, to $\mathcal{G}$ for some $c = (c_{(1)}, c_{(2)}, c_{(3)})$ returned by $\mathsf{SIM}_{\mathsf{NCE}}$. Let the probability of this event be ϵ_l. One can now derive that $\mathcal{B}$ can invert f with probability $\frac{\epsilon_l}{l}$, which is non-negligible, if ϵ_l is non-negligible. This is a contradiction to the assumption that the element c cannot be inverted with noticeable probability. What follows is:

$$\left|\mathcal{P}_7 - \mathcal{P}_8\right| \leq \sum_{i=0}^{q} \frac{\epsilon_i}{q}$$

Clearly, q is at most polynomial, and therefore the given sum is also negligible.

Game 9: Finally, RECV-SIM-ideal is run with the simulator given in Game 8. This is only an internal change: the adversary does not note any difference:

$$\left|\mathcal{P}_8 - \mathcal{P}_9\right| = 0$$

As an upper bound, one can therefore derive:

$$\left|\mathcal{P}_0 - \mathcal{P}_9\right| \leq \sum_{i=1}^{9} \left|\mathcal{P}_{i-1} - \mathcal{P}_i\right|$$

which is negligible. Clearly this also means that:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM\text{-}real}}_{\mathsf{ENC},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM\text{-}ideal}}_{\mathsf{ENC},\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}}}(\lambda) = 1]\right|$$

is also negligible, which proves the Theorem. □

Appendix G

Virtual Smart-Cards: Detailed Experimental Results

The Pass2Sign scheme (i.e., the one with message blindness) was implemented to have performance figures demonstrating its practicality. Summarized, setup with a decent security parameter (4,096 Bit Moduli for both the non-committing encryption and the signatures) takes roughly 20 seconds, while for each signature generation the non-optimized implementation takes far less than two seconds. In a real-life deployment, this performance is more than sufficient for most use-cases.

Setting. Measured were the setup and sign protocol with three different RSA-moduli sizes, 1,024, 2,048, and 4,096 Bit to account for different security requirements. The key size is used for both the signing key and the RSA trapdoor permutation in the non-committing encryption scheme. To instantiate the random oracles $\mathcal{K}, \mathcal{G}$, and $\mathcal{H}$ SHA-512 is used. Each call was prefixed accordingly. The instantiation of the full-domain hash $\mathcal{H}_{\mathsf{RSA}}$ is based on the construction given in [BR93], and uses rejection sampling to uniformly map into $\mathbb{Z}_N^*$. The pseudo-code is depicted in Figure 1, based on the ideas given by Frädrich et al. [FPP+16].

Data: Modulus N, a hash-function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$, a message $m \in \{0,1\}^*$
Result: A hash h uniformly distributed in $\mathbb{Z}_N^*$
$c \leftarrow 0$;
$h \leftarrow \perp$;
$s \leftarrow \lfloor((|N| - 1)/\lambda)\rfloor + 1$; //$|N|$ is the bit-length of N
while *true* **do**
  $h \leftarrow \mathcal{H}(N, m, c)||\mathcal{H}(N, m, c+1)|| \ldots ||\mathcal{H}(N, m, c+s-1)$;
  $h \leftarrow h \gg (s \cdot \lambda - |N|)$; //shift out excessive bits
  if $\gcd(h, N) = 1 \wedge h \leq N$ *//h is interpreted as a natural number* **then**
    return h;
  end
  $c \leftarrow c + 1$;
end

Algorithm 1: Pseudocode of the FDH-Implementation

The implementation uses Java 8. The server was run on a Intel i7-3740QM with 2.7GHz

and 16GB RAM, while the device was a Nexus 10 with 1.7GHz, 2GB RAM and Android 5.1.1. Not implemented were any optimizations such as multi-threading, RSA-CRT, connection pooling, or keep-alive of connections. The communication partners send messages using standard TCP-Sockets, and open a new connection for each new sub-protocol. This time is included in the measurements. $\mathcal{F}_{\mathsf{Auth}}$ has been implemented using digital signatures with pre-shared keys. Calls to $\mathcal{F}_{\mathsf{CA}}$ are included in the measurements. However, the measurements do not contain network round-trip times (typically between 20ms and 400ms), as these clearly depend on the current locations of the server and the user, and only add an additional constant to the bare run-time measurements. For example, assuming a round-trip time of 100ms, one can roughly add these 100ms to the combined run-time.

As the timings should focus on the protocol, they do not include the generation of the setup parameters such as the keys for the non-comitting encryption scheme, or the generation of the session and query identifiers *sid*, *qid*.

For all the following confidence intervals and tables 100 runs were measured.

Setup Protocol. For measuring the setup protocol, the password that is an input to the protocol is a fixed string. The runtime box-plots are in Figure G.2 for the device and Figure G.1 for the server. The corresponding percentiles are depicted in Table G.1.

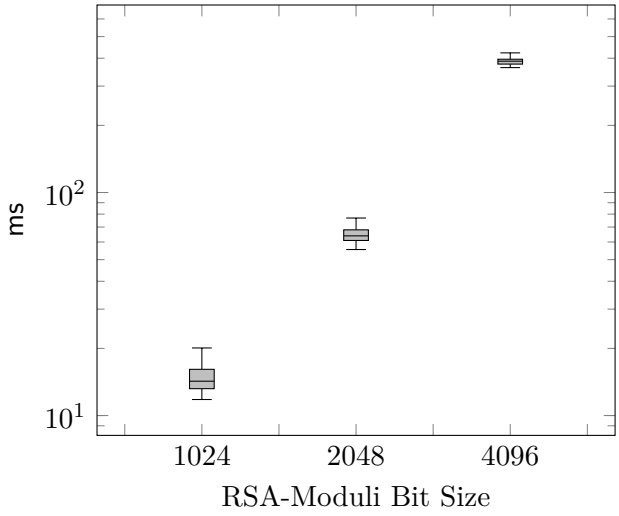

Figure G.1: Box-Plots of the setup protocol measurements in ms for the server

As it can be seen, the time for the setup on the device varies a lot, which results from the randomized RSA-key generation algorithm. Compared with the key generation on smart-cards, the protocol (including communication time) is still significantly faster though. For a 2,048 Bit modulus, even rather powerful smart-cards take more than one minute for an RSA key pair generation[1], whereas the full setup protocol would take around four seconds on average including network delay.

Signing Protocol. For each signing request, the correct password was used, i.e., no "failed" signing attempts were measured. Additionally to the *sid* now also a query identifier *qid* is

[1] `http://www.pronew.com.tw/download/doc/400SmartCard080907.pdf` (last visit: April 19, 2018)

Table G.1: Percentiles for setup in milliseconds for the device and server

	Device			Server		
	1,024Bit	2,048Bit	4,096Bit	1,024Bit	2,048Bit	4,096Bit
Min.:	180.82	398.27	8′238.22	11.80	55.50	363.10
25%:	443.77	2′063.02	10′760.03	13.22	61.07	376.53
Median:	648.11	3′335.34	14′343.46	14.33	63.96	388.10
75%:	1′130.72	4′700.18	20′913.41	16.14	68.22	396.01
90%:	1′527.36	6′629.75	37′721.16	17.23	74.40	428.47
95%:	2′272.43	7′876.96	40′008.34	19.85	83.44	444.75
Max.:	3′927.65	14′649.61	57′822.42	48.92	99.93	478.49
Average:	855.58	3′646.27	16′202.58	15.20	65.69	393.27

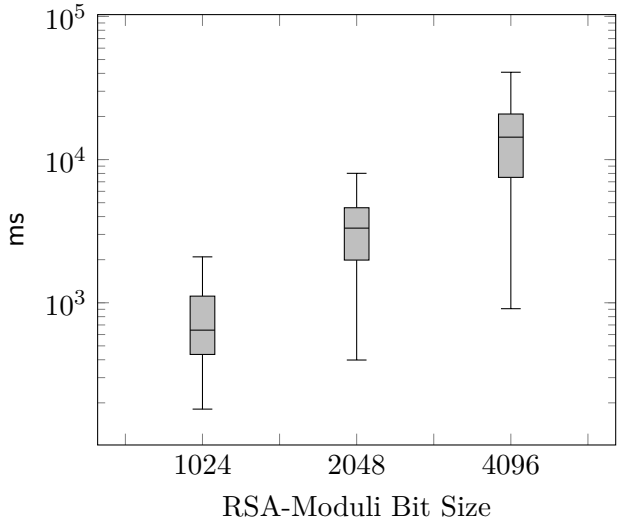

Figure G.2: Box-Plots of the setup protocol measurements in ms for the device

Table G.2: Percentiles for signing in milliseconds for the device and the server

	Device			Server		
	1,024Bit	2,048Bit	4,096Bit	1,024Bit	2,048Bit	4,096Bit
Min.:	16.49	73.47	470.30	10.90	62.43	452.01
25%:	18.06	78.44	478.84	11.32	63.56	454.08
Median:	19.08	79.83	482.61	11.76	64.53	456.38
75%:	20.64	82.38	487.69	12.60	66.58	470.95
90%:	22.25	92.19	671.13	14.41	68.39	501.75
95%:	23.91	113.66	883.83	14.96	72.06	521.64
Max.:	42.80	138.18	4′994.40	27.77	78.98	578.58
Average:	19.79	83.40	574.41	12.31	65.50	466.73

given as input, which is assumed to be generated by an external protocol. The box-plots are in Figure G.3 and Figure G.4. The corresponding percentiles are depicted in Table G.2.

The figures show that the time required for signing is — from a practical perspective —

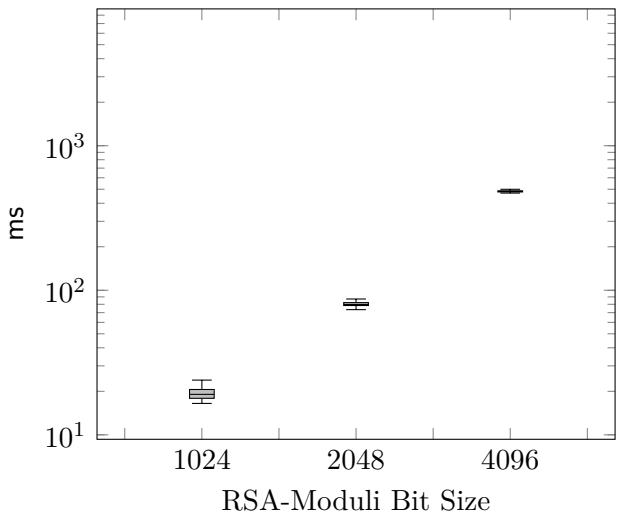

Figure G.3: Box-Plots of the signing protocol measurements in ms for the device

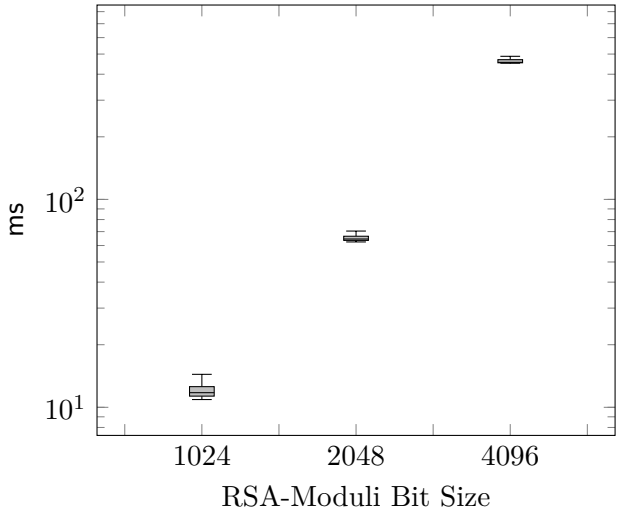

Figure G.4: Box-Plots of the signing protocol measurements in ms for the server

nearly constant for each security parameter. Including the communication overhead, the protocol is still faster than a real smart-card, which requires at least one second for signing with a 2,048 Bit key.

Appendix H

Nielsen's Construction

The construction of the underlying encryption scheme by Nielsen looks very similar to the one given in Chapter 5. However, it comes without labels and without the "MAC" $c_{(3)}$. The message and party IDs are omitted to focus on the construction itself, which essentially the well-known IND-CPA-secure Bellare-Rogaway encryption-scheme [BR93].

Construction H.1 (ENC)**.** *Their encryption scheme is given as follows.*

$\mathsf{KeyGen}_{\mathsf{ENC}}(1^\lambda)$: *Generate a TDP key pair, i.e.,* $(f, f^{-1}, \Sigma) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{TDP}}(1^\lambda)$*. Set* $\mathcal{MS} = \{0,1\}^\lambda$*. Output the public key* $\mathsf{pk}_{\mathsf{ENC}} = (f, \Sigma)$*, and* $\mathsf{sk}_{\mathsf{ENC}} = f^{-1}$ *as the secret key.*

$\mathsf{Enc}_{\mathsf{ENC}}(\mathsf{pk}_{\mathsf{ENC}}, m)$: *Draw* $x \xleftarrow{\$} \mathsf{Sample}_{\mathsf{TDP}}(1^\lambda)$*, and compute* $c_{(1)} \leftarrow f(x)$*, and* $c_2 \leftarrow \mathcal{H}(x) \oplus m$*. Output the ciphertext* $c \leftarrow (c_{(1)}, c_{(2)})$*.*

$\mathsf{Dec}_{\mathsf{ENC}}(\mathsf{sk}_{\mathsf{ENC}}, c)$: *Parse* c *as* $(c_{(1)}, c_{(2)})$*. Compute* $x' \leftarrow f^{-1}(c_{(1)})$*, and* $m' \leftarrow \mathcal{H}(x') \oplus c_{(2)}$*. Output* m'*.*

Nielsen's Secure Message Transfer $\mathcal{F}_{\mathsf{SMT}}$. As usual, $\mathcal{L}$ returns the length of a message m. Depicted in Figure H.1 is the plain definition given by Nielsen [Nie02]. Note, this definition does not give away any ciphertext to the environment.

1. `Send.` Upon input $(\mathsf{SEND}, mid, j, m)$ of party $\mathcal{P}_i$ do:
   - Deliver $(\mathsf{RECEIVE}, mid, i, m)$ to $\mathcal{P}_j$, and send $(\mathsf{RECEIVE}, mid, i, \mathcal{L}(m))$ to $\mathcal{A}$.

Figure H.1: Nielsen's Secure Message Transmission functionality

Appendix I

The TAG-Based Chameleon-Hash by Brzuska et al.

Here, the construction by Brzuska et al. [BFF+09] is restated, in their model. A reformulation in the given framework is straightforward.

Construction I.1 (CH). *Let $\mathcal{H}_x : \{0,1\}^* \to \mathbb{Z}_x^*$, $x \in \mathbb{N}$, denote a random oracle. Now, their chameleon-hash is defined as follows:*

$\mathsf{PPGen}_{\mathsf{CH}}$. *The algorithm $\mathsf{PPGen}_{\mathsf{CH}}$ generates the public parameters in the following way:*

1. *Return $\emptyset$.*

$\mathsf{KeyGen}_{\mathsf{CH}}$. *The algorithm $\mathsf{KeyGen}_{\mathsf{CH}}$ generates the key pair in the following way:*

1. *Generate two primes p and q using $\mathsf{RSAGen}(1^\lambda)$, which also outputs e coprime to $\varphi(N)$. Set $\mathsf{sk}_{\mathsf{CH}} = (p,q)$. Let $N = pq$. Set $\mathsf{pk}_{\mathsf{CH}} = (N,e)$.*
2. *Return $(\mathsf{pk}_{\mathsf{CH}}, \mathsf{sk}_{\mathsf{CH}})$.*

$\mathsf{Hash}_{\mathsf{CH}}$. *To hash a message m with a tag τ, randomness r w.r.t. $\mathsf{pk}_{\mathsf{CH}} = (N,e)$ do:*

1. *Let $g \leftarrow \mathcal{H}_N(\tau, m)$.*
2. *Let $h \leftarrow g r^e \bmod N$.*
3. *Return h.*

$\mathsf{Adapt}_{\mathsf{CH}}$. *To find a collision w.r.t. m with tags τ, τ', m', randomness r, and $\mathsf{sk}_{\mathsf{CH}}$ do:*

1. *Compute $g \leftarrow \mathcal{H}_N(\tau, m)$, and $h \leftarrow g r^e \bmod N$.*
2. *Compute $g' \leftarrow \mathcal{H}_N(\tau', m')$ and $r' \leftarrow (h(g'^{-1}))^d \bmod N$.*
3. *Return r'.*

The proof of security (in a less strict model) is given in the original paper by Brzuska et al. [BFF+09].

Appendix J

Additional Security Properties for SSS

Here, some additional definitions for SSS are given, which are not needed for the main part of this thesis. Namely, these are unlinkability and non-interactive public accountability.

Unlinkability. Unlinkability prohibits an adversary to decide how a signature was generated, i.e., from which signature a sanitized signature was derived. This is the stronger definition by Brzuska et al. [BPS13], where even the signer can be malicious. This game is similar to privacy with the same constraints. Namely, compared to the privacy game, the adversary can also input the signatures to be used. It receives full oracle access, while the signing, and the proof oracles, can be simulated by the adversary.

Experiment $\mathsf{Unlinkability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \leftarrow \{0,1\}$
$a \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}, \mathcal{O}^{\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}},b)}}(\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
where oracle LoRSanit on input of $m_0, \mathsf{MOD}_0, \sigma_0, m_1, \mathsf{MOD}_1, \sigma_1, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, b$:
return $\perp$, if $\mathsf{ADM}_0 \neq \mathsf{ADM}_1 \vee \mathsf{MOD}_0(m_0) \neq \mathsf{MOD}_1(m_1) \vee$
$\mathsf{ADM}_0(\mathsf{MOD}_0) \neq \mathsf{ADM}_1(\mathsf{MOD}_1) \vee$
$\mathsf{Verify}_{\mathsf{SSS}}(m_0, \sigma_0, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) \neq \mathsf{Verify}_{\mathsf{SSS}}(m_1, \sigma_1, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
return $(m', \sigma') \leftarrow \mathsf{Sanit}_{\mathsf{SSS}}(m_b, \mathsf{MOD}_b, \sigma_b, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})$
return 1, if $a = b$
return 0

Figure J.1: SSS Unlinkability

Definition J.1 (Unlinkability)**.** *An* SSS *is unlinkable, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\left|\Pr[\mathsf{Unlinkability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure J.1.

Non-Interactive Public Accountability. Non-interactive public accountability allows everyone to decide whether a sanitizer was involved. This is modeled by requiring that $\mathsf{Judge}_{\mathsf{SSS}}$ works with an empty proof, i.e., $\pi = \perp$. Hence, no secret keys are required to find the accountable party.

Experiment $\mathsf{Pub\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{SSS}} \xleftarrow{\$} \mathsf{PPGen}_{\mathsf{SSS}}(1^\lambda)$
$(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{Sig}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathsf{San}}_{\mathsf{SSS}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{pk}^*, m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_{\mathsf{SSS}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{SSS}},\cdot,\cdot)}, \mathcal{O}^{\mathsf{Sanit}_{\mathsf{SSS}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{San}}_{\mathsf{SSS}})}}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{ADM}_i, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i})$ and σ_i
index the queries/answers to/from $\mathsf{Sign}_{\mathsf{SSS}}$
for $j = 1, 2, \ldots, q'$ let $(m_j, \mathsf{MOD}_j, \sigma_j, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},j})$ and (m'_j, σ'_j)
index the queries/answers to/from $\mathsf{Sanit}_{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^*) = \mathsf{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS},i}, m_i, \sigma_i) \wedge$
$\mathsf{Judge}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS}}, \mathsf{pk}^*, \perp) = \mathsf{Signer}$
return 1, if $\mathsf{Verify}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}) = \mathsf{true} \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{SSS},j}, m'_j, \sigma'_j) \wedge$
$\mathsf{Judge}_{\mathsf{SSS}}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}^{\mathsf{San}}_{\mathsf{SSS}}, \perp) = \mathsf{Sanitizer}$
return 0

Figure J.2: SSS Public Accountability

Definition J.2 (Non-Interactive Public Accountability)**.** *An* SSS *is non-interactive publicly accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathsf{Pub\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$$

where the corresponding experiment is defined in Figure J.2.